LECTURE NOTES

ON

DATABASE MANAGEMENT SYSTEMS

B.TECH II YEAR I SEMESTER

(JNTUA-R15)

Ms. BRINDHA KUNDAVARAN M. Tech.,

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

S. V. ENGINEERING COLLEGE FOR WOMEN

KARAKAMBADI ROAD, OPP. LIC TRAINING CENTER, TIRUPATI (A.P) - 517507

UNIT-1

UNIT-1

INTRODUCTION TO BASIC CONCEPTS OF DATABASE SYSTEMS:

What is Data?

The raw facts are called as data. The word "raw" indicates that they have not been processed.

Ex: For example 89 is the data.

What is information?

The processed data is known as information.

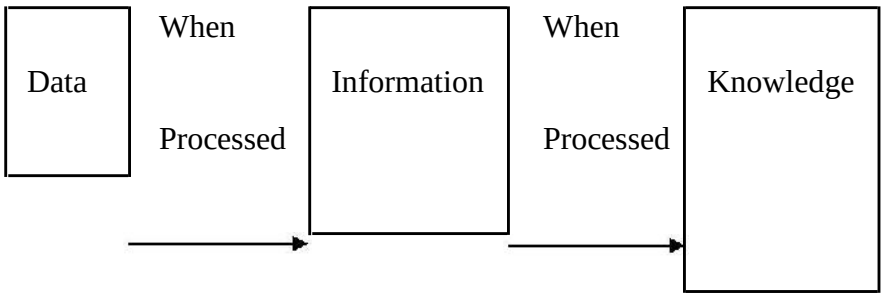
Ex: Marks: 89; then it becomes information.

What is Knowledge?

- Knowledge refers to the practical use of information.
- Knowledge necessarily involves a personal experience.

DATA/INFORMATION PROCESSING:

The process of converting the data (raw facts) into meaningful information is called as data/information processing.



Note: In business processing knowledge is more useful to make decisions for any organization.

DIFFERENCE BETWEEN DATA AND INFORMATION:

DATA	INFORMATION
1.Raw facts	1.Processed data
2. It is in unorganized form	2. It is in organized form
3. Data doesn't help in decision making process	3. Information helps in decision making process

FILE ORIENTED APPROACH:

The earliest business computer systems were used to process business records and produce information. They were generally faster and more accurate than equivalent manual systems. These systems stored groups of records in separate files, and so they were called **file processing systems.**

1. File system is a collection of data. Any management with the file system, user has to write the procedures

2. File system gives the details of the data representation and Storage of data.

3. In File system storing and retrieving of data cannot be done efficiently.

4. Concurrent access to the data in the file system has many problems like a Reading the file while other deleting some information, updating some information

5. File system doesn't provide crash recovery mechanism.

Eg. While we are entering some data into the file if System crashes then content of the file is lost.

6. Protecting a file under file system is very difficult.

The typical file-oriented system is supported by a conventional operating system. Permanent records are stored in various files and a number of different application programs are written to extract records from and add records to the appropriate files.

DISADVANTAGES OF FILE-ORIENTED SYSTEM:

The following are the disadvantages of File-Oriented System:

Data Redundancy and Inconsistency:

Since files and application programs are created by different programmers over a long period of time, the files are likely to be having different formats and the programs may be written in several programming languages. Moreover, the same piece of information may be duplicated in several places. This redundancy leads to higher storage and access cost. In addition, it may lead to data inconsistency.

Difficulty in Accessing Data:

The conventional file processing environments do not allow needed data to be retrieved in a convenient and efficient manner. Better data retrieval system must be developed for general use.

Data Isolation:

Since data is scattered in various files, and files may be in different formats, it is difficult to write new

application programs to retrieve the appropriate data.

Concurrent Access Anomalies:

In order to improve the overall performance of the system and obtain a faster response time, many systems allow multiple users to update the data simultaneously. In such an environment, interaction of concurrent updates may result in inconsistent data.

Security Problems:

Not every user of the database system should be able to access all the data. For example, in banking system, payroll personnel need only that part of the database that has information about various bank employees. They do not need access to information about customer accounts. It is difficult to enforce such security constraints.

Integrity Problems:

The data values stored in the database must satisfy certain types of consistency constraints. For example, the balance of a bank account may never fall below a prescribed amount. These constraints are enforced in the system by adding appropriate code in the various application programs. When new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items for different files.

Atomicity Problem:

A computer system like any other mechanical or electrical device is subject to failure. In many applications, it is crucial to ensure that once a failure has occurred and has been detected, the data are restored to the consistent state existed prior to the failure.

Example:

Consider part of a savings-bank enterprise that keeps information about all customers and savings accounts. One way to keep the information on a computer is to store it in operating system files. To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including:

• A program to debit or credit an account

• A program to add a new account

• A program to find the balance of an account
• A program to generate monthly statements

Programmers wrote these application programs to meet the needs of the bank. New application programs are added to the system as the need arises. For example, suppose that the savings bank decides to offer checking accounts. As a result, the bank creates new permanent files that contain information about all the checking accounts maintained in the bank, and it may have to write new application programs to deal with situations that do not arise in savings accounts, such as overdrafts. Thus, as time goes by, the system acquires more files and more application programs. The system stores permanent records in various files, and it needs different Application programs to extract records from, and add records to, the appropriate files. Before database management systems (DBMS) came along, organizations usually stored information in such systems.

Organizational information in a file-processing system has a number of major disadvantages:

1. Data Redundancy and Inconsistency:

The address and telephone number of a particular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records. This redundancy leads to higher storage and access cost. In, it may lead to data inconsistency; that is, the various copies of the same data may no longer agree. For example, a changed customer address may be reflected in savings-account records but not elsewhere in the system.

2. **Difficulty in Accessing Data:**

Suppose that one of the bank officers needs to find out the names of all customers who live within a particular postal-code area. The officer asks the data-processing department to generate such a list. Because there is no application program to generate that. The bank officer has now two choices: either obtain the list of all customers and extract the needed information manually or ask a system programmer to write the necessary application program. Both alternatives are obviously unsatisfactory.

3. **Data Isolation:**

Because data are scattered in various files and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

4. **Integrity Problems:**

The balance of a bank account may never fall below a prescribed amount (say, $25). Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.

5. **Atomicity Problems:**

A computer system, like any other mechanical or electrical device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer $50 from account *A* to account *B*. If a system failure occurs during the execution of the program, it is possible that the $50 was removed from account *A* but was not credited to account *B*, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be *atomic*—it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system.

6. **Concurrent-Access Anomalies:**

For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. In such an environment, interaction of concurrent updates may result in inconsistent data. Consider bank account *A*, containing $500. If two customers withdraw funds (say $50 and $100 respectively) from account *A* at about the same time, the result of the concurrent executions may leave the account in an incorrect (or inconsistent) state. Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce that value by the amount being withdrawn, and write the result back. If the two programs run concurrently, they may both read the value $500, and write back $450 and $400, respectively. Depending on which one writes the value last, the account may contain $450 or $400, rather than the correct value of $350. To guard against this possibility, the system must maintain some form of supervision. But supervision is difficult to provide because data may be accessed by many different application programs that have not been coordinated previously.

7. **Security Problems:**

Not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about the various bank employees. They do not need access to information about customer accounts. But, since application programs are added to the system in an ad hoc manner, enforcing such security constraints is difficult. These difficulties, among others, prompted the development of database systems.

INTRODUCTION TO DATABASES:

History of Database Systems:

1950s and early 1960s:

• Magnetic tapes were developed for data storage

• Data processing tasks such as payroll were automated, with data stored on tapes.

• Data could also be input from punched card decks, and output to printers.

• Late 1960s and 1970s: The use of hard disks in the late 1960s changed the scenario for data processing

greatly, since hard disks allowed direct access to data.

• With disks, network and hierarchical databases could be created that allowed data structures such as lists and trees to be stored on disk. Programmers could construct and manipulate these data structures.

• With disks, network and hierarchical databases could be created that allowed data structures such as lists and trees to be stored on disk. Programmers could construct and manipulate these data structures.

• In the 1970's the EF CODD defined the **Relational Model.**

Inthe1980's:

• Initial commercial relational database systems, such as IBM DB2, Oracle, Ingress, and DEC Rdb, played a major role in advancing techniques for efficient processing of declarative queries.

• In the early 1980s, relational databases had become competitive with network and hierarchical database systems even in the area of performance.

• The 1980s also saw much research on parallel and distributed databases, as well as initial work on object-oriented databases.

Early 1990s:

• The SQL language was designed primarily in the 1990's.

• And this is used for the transaction processing applications.

• Decision support and querying re-emerged as a major application area for databases.

• Database vendors also began to add object-relational support to their databases.

Late 1990s:

• The major event was the explosive growth of the World Wide Web.

• Databases were deployed much more extensively than ever before. Database systems now had to support very high transaction processing rates, as well as very high reliability and 24 * 7 availability (availability 24 hours a day, 7 days a week, meaning no downtime for scheduled maintenance activities).

• Database systems also had to support Web interfaces to data.

The Evolution of Database systems:

The Evolution of Database systems are as follows:

1. File Management System
2. Hierarchical database System
3. Network Database System
4. Relational Database System

File Management System:

The file management system also called as FMS in short is one in which all data is stored on a single large file. The main disadvantage in this system is searching a record or data takes a long time. This lead to the introduction of the concept, of indexing in this system. Then also the FMS system had lot of drawbacks to name a few like updating or modifications to the data cannot be handled easily, sorting the records took long time and so on. All these drawbacks led to the introduction of the Hierarchical Database System.

Hierarchical Database System:

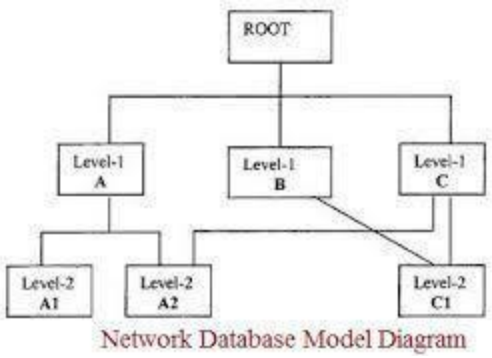
The previous system FMS drawback of accessing records and sorting records which took a long time was removed in this by the introduction of parent-child relationship between records in database. The origin of the data is called the root from which several branches have data at different levels and the last level is called the

leaf. The main drawback in this was if there is any modification or addition made to the structure then the whole structure needed alteration which made the task a tedious one. In order to avoid this next system took its origin which is called as the Network Database System.

Fig: Hierarchical Database System

Network Database System:

In this the main concept of many-many relationships got introduced. But this also followed the same technology of pointers to define relationships with a difference in this made in the introduction if grouping of data items as sets.



Network Database Model Diagram

Relational Database System:

In order to overcome all the drawbacks of the previous systems, the Relational Database System got introduced in which data get organized as tables and each record forms a row with many fields or attributes in it. Relationships between tables are also formed in this system.

Name	FName	City	Age	Salary
Smith	John	3	35	$280
Doe	Jane	1	28	$325
Brown	Scott	3	41	$265
Howard	Shemp	4	48	$359
Taylor	Tom	2	22	$250

DATABASE:

A database is a collection of related data.

(OR)

A database is a collection of information that is organized so that it can be easily accessed, managed and updated.

Examples / Applications of Database Systems:

The following are the various kinds of applications/organizations uses databases for their business processing activities in their day-to-day life. They are:

1. **Banking:** For customer information, accounts, and loans, and banking transactions.

2. **Airlines:** For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner—terminals situated around the world accessed the central database system through phone lines and other data networks.

3. **Universities:** For student information, course registrations, and grades.

4. **Credit Card Transactions:** For purchases on credit cards and generation of monthly statements.

5. **Telecommunication:** For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

6. **Finance:** For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds.

7. **Sales:** For customer, product, and purchase information.

8. **Manufacturing:** For management of supply chain and for tracking production of items in factories, inventories of items in warehouses/stores, and orders for items.

9. **Human resources:** For information about employees, salaries, payroll taxes and benefits, and for generation of paychecks.

10. **Railway Reservation Systems:** For reservations and schedule information.

11. **Web:** For access the Back accounts and to get the balance amount.

12. **E –Commerce:** For Buying a book or music CD and browse for things like watches, mobiles from the Internet.

CHARACTERISTICS OF DATABASE:

The database approach has some very characteristic features which are discussed in detail below:

Structured and Described Data:

Fundamental feature of the database approach is that the database system does not only contain the data but also the complete definition and description of these data. These descriptions are basically details about the extent, the structure, the type and the format of all data and, additionally, the relationship between the data. This kind of stored data is called metadata ("data about data").

Separation of Data and Applications:

Application software does not need any knowledge about the physical data storage like encoding, format, storage place, etc. It only communicates with the management system of a database (DBMS) via a standardized interface with the help of a standardized language like SQL. The access to the data and the metadata is entirely done by the DBMS. In this way all the applications can be totally separated from the data.

Data Integrity:

Data integrity is a byword for the quality and the reliability of the data of a database system. In a broader sense data integrity includes also the protection of the database from unauthorized access (confidentiality) and unauthorized changes. Data reflect facts of the real world.

Transactions:

A transaction is a bundle of actions which are done within a database to bring it from one consistent state to a new consistent state. In between the data are inevitable inconsistent. *A* transaction is atomic what

means that it cannot be divided up any further. Within a transaction all or none of the actions need to be carried out. Doing only a part of the actions would lead to an inconsistent database state.

Example: One example of a transaction is the transfer of an amount of money from one bank account to another.

Data Persistence:

Data persistence means that in a DBMS all data is maintained as long as it is not deleted explicitly. The life span of data needs to be determined directly or indirectly be the user and must not be dependent on system features. Additionally data once stored in a database must not be lost. Changes of a database which are done by a transaction are persistent. When a transaction is finished even a system crash cannot put the data in danger

TYPES OF DATABASES:

- Database can be classified according to the following factors. They are:

1. Number of Users
2. Database Location
3. Expected type
4. Extent of use

1. Based on number of Users:

- According to the number of users the databases can be classified into following types. They are :

a). Single user b). Multiuser

Single user database:

- Single user database supports only one user at a time.
- Desktop or personal computer database is an example for single user database.

Multiuser database:

- Multi user database supports multiple users at the same time.

- Workgroup database and enterprise databases are examples for multiuser database.

Workgroup database:

If the multiuser database supports relatively small number of users (fewer than 50) within an organization is called as Workgroup database.

Enterprise database:

If the database is used by the entire organization and supports multiple users (more than 50) across many departments is called as Enterprise database.

2. Based on Location:

According to the location of database the databases can be classified into following types. They are:

a). Centralized Database b). Distributed Database

Centralized Database:

It is a database that is located, stored, and maintained in a single location. This location is most often a central computer or database system, for example a desktop or server CPU, or a mainframe computer. In most cases, a centralized database would be used by an organization (e.g. a business company) or an institution (e.g. a university.)

Distributed Database:

A distributed database is a database in which storage devices are not all attached to a common CPU. It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers.

INTRODUCTION TO DATABASE-MANAGEMENT SYSTEM:

Database Management System:

- A **database-management system** (DBMS) is a collection of interrelated data and a set of programs to access those data.

- The DBMS is a general purpose software system that facilitates the process of defining constructing and manipulating databases for various applications.

Goals of DBMS:

The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*

1. Manage large bodies of information

2. Provide convenient and efficient ways to store and access information

3. Secure information against system failure or tampering

4. Permit data to be shared among multiple users

Properties of DBMS:

1. A Database represents some aspect of the real world. Changes to the real world reflected in the database.

2. A Database is a logically coherent collection of data with some inherent meaning.

3. A Database is designed and populated with data for a specific purpose.

Need of DBMS:

1. Before the advent of DBMS, organizations typically stored information using a "File Processing Systems".

Example of such systems is File Handling in High Level Languages like C, Basic and COBOL etc., these systems have Major disadvantages to perform the Data Manipulation. So to overcome those drawbacks now we are using the DBMS.

2. Database systems are designed to manage large bodies of information.

3. In addition to that the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.

ADVANTAGES OF A DBMS OVER FILE SYSTEM:

Using a DBMS to manage data has many advantages:

Data Independence:

Application programs should be as independent as possible from details of data representation and storage. The DBMS can provide an abstract view of the data to insulate application code from such details.

Efficient Data Access:

A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. This feature is especially important if the data is stored on external storage devices.

Data Integrity and Security:

If data is always accessed through the DBMS, the DBMS can enforce integrity constraints on the data. For example, before inserting salary information for an employee, the DBMS can check that the department budget is not exceeded. Also, the DBMS can enforce *access controls* that govern what data is visible to different classes of users.

Concurrent Access and Crash Recovery:

A database system allows several users to access the database concurrently. Answering different questions from different users with the same (base) data is a central aspect of an information system. Such concurrent use of data increases the economy of a system.

An example for concurrent use is the travel database of a bigger travel agency. The employees of different branches can access the database concurrently and book journeys for their clients. Each travel agent sees on his interface if there are still seats available for a specific journey or if it is already fully booked.

A DBMS also protects data from failures such as power failures and crashes etc. by the recovery schemes such as backup mechanisms and log files etc.

DATA ADMINISTRATION:

When several users share the data, centralizing the administration of data can offer significant improvements. Experienced professionals, who understand the nature of the data being managed, and how different groups of users use it, can be responsible for organizing the data representation to minimize redundancy and fine-tuning the storage of the data to make retrieval efficient.

Reduced Application Development Time:

DBMS supports many important functions that are common to many applications accessing data stored in the DBMS. This, in conjunction with the high-level interface to the data, facilitates quick development of applications. Such applications are also likely to be more robust than applications developed from scratch because many important tasks are handled by the DBMS instead of being implemented by the application.

DISADVANTAGES OF DBMS:

Danger of a Overkill:

For small and simple applications for single users a database system is often not advisable.

Complexity:

A database system creates additional complexity and requirements. The supply and operation of a database management system with several users and databases is quite costly and demanding.

Qualified Personnel:

`The professional operation of a database system requires appropriately trained staff. Without a qualified database administrator nothing will work for long.

Costs:

Through the use of a database system new costs are generated for the system itself but also for additional hardware and the more complex handling of the system.

Lower Efficiency:

A database system is a multi-use software which is often less efficient than specialized software which is produced and optimized exactly for one problem.

DATABASE USERS & DATABASE ADMINISTRATORS:

People who work with a database can be categorized as database users or database administrators.

Database Users:

There are four different types of database-system users, differentiated by the way they expect to interact with the system.

Naive users:

Naive users are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.

For example, a bank teller who needs to transfer $50 from account *A* to account *B* invokes a program called *transfer*. This program asks the teller for the amount of money to be transferred, the account from which the money is to be transferred, and the account to which the money is to be transferred.

Application programmers:

Application programmers are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. **Rapid application development (RAD)**

tools are tools that enable an application programmer to construct forms and reports without writing a program.

Sophisticated users:

Sophisticated users interact with the system without writing programs. Instead, they form their requests in a database query language. They submit each such query to a **query processor**, whose function is to break down DML statements into instructions that the storage manager understands. Analysts who submit queries to explore data in the database fall in this category.

Specialized users:

Specialized users are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework.

Database Administrator:

One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a **database administrator** (**DBA**).

Database Administrator Functions/Roles:

The functions of a DBA include:

Schema definition:

The DBA creates the original database schema by executing a set of data definition statements in the DDL, Storage structure and access-method definition.

Schema and physical-organization modification:

The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.

Granting of authorization for data access:

By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.

Routine maintenance:

Examples of the database administrator's routine maintenance activities are:

1. Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.

2. Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required.

3. Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

LEVELS OF ABSTRACTION IN A DBMS:

Hiding certain details of how the data are stored and maintained. A major purpose of database system is to provide users with an "Abstract View" of the data. In DBMS there are 3 levels of data abstraction. The goal of the abstraction in the DBMS is to separate the users request and the physical storage of data in the database.

Levels of Abstraction:

Physical Level:

- The lowest Level of Abstraction describes "How" the data are actually stored.
- The physical level describes complex low level data structures in detail.

Logical Level:

- This level of data Abstraction describes "What" data are to be stored in the database and what relationships exist among those data.

- Database Administrators use the logical level of abstraction.

View Level:

- It is the highest level of data Abstracts that describes only part of entire database.
- Different users require different types of data elements from each database.
- The system may provide many views for the some database.

THREE SCHEMA ARCHITECTURE:

Schema:

The overall design of the database is called the "Schema" or "Meta Data". A database schema corresponds to the programming language type definition. The value of a variable in programming language corresponds to an "Instance" of a database Schema.

Three Schema Architecture:

The goal of this architecture is to separate the user applications and the physical database. In this architecture, schemas can be defined at the following three levels:

1. The **internal level** has an **internal schema,** which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.

2. The **conceptual level** has a **conceptual schema,** which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. A high-level data model or an implementation data model can be used at this level.

3. The **external** or **view level** includes a number of **external schemas** or **user views.** Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. A high-level data model or an implementation data model can be used at this level.

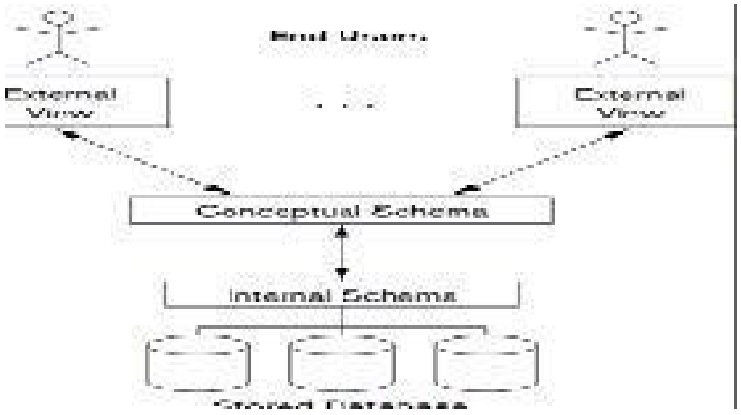


Fig: Three-Schema Architecture

DATA INDEPENDENCE:

- A very important advantage of using DBMS is that it offers Data Independence.

- The ability to modify a scheme definition in one level without affecting a scheme definition in a higher level is called **data independence**.

- There are two kinds:

1. Physical Data Independence
2. Logical Data Independence

Physical Data Independence:

- The ability to modify the physical schema without causing application programs to be rewritten
- Modifications at this level are usually to improve performance.

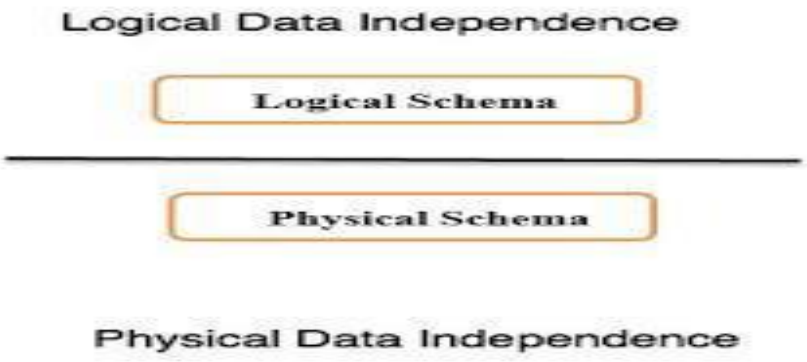


Fig: Data Independence

Logical Data Independence:

- The ability to modify the conceptual schema without causing application programs to be rewritten
- Usually done when logical structure of database is altered

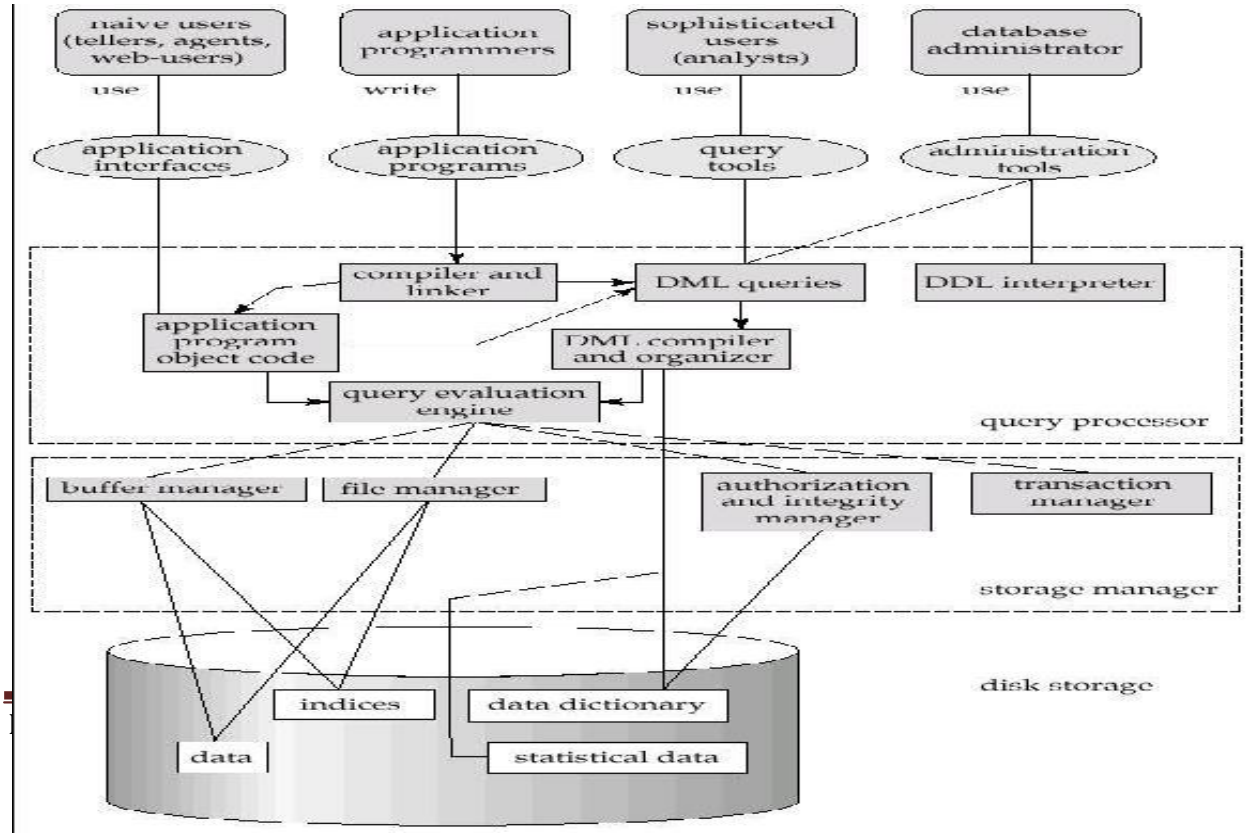
- Logical data independence is harder to achieve as the application programs are usually heavily dependent on the logical structure of the data.

DATABASE SYSTEM STRUCTURE:

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the storage manager and the query processor components.

The storage manager is important because databases typically require a large amount of storage space. Some Big organizations Database ranges from Giga bytes to Tera bytes. So the main memory of computers cannot store this much information, the information is stored on disks. Data are moved between disk storage and main memory as needed.

The query processor also very important because it helps the database system simplify and facilitate access to data. So quick processing of updates and queries is important. It is the job of the database system to translate updates and queries written in a nonprocedural language,



Storage Manager:

A storage manager is a program module that provides the interface between the low level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file manager. The storage manager translates the various DML statements into low-level file-system commands. Thus, the storage manager is responsible for storing, retrieving, and updating data in the database.

Storage Manager Components:

Authorization and integrity manager which tests for the satisfaction of integrity constraints and checks the authority of users to access data.

Transaction manager which ensures that the database itself remains in a consistent state despite system failures, and that concurrent transaction executions proceed without conflicting.

File manager: which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.

Buffer manager which is responsible for fetching data from disk storage into main memory. Storage manager implements several data structures as part of the physical system implementation. Data files are used to store the database itself. Data dictionary is used to stores metadata about the structure of the database, in particular the schema of the database.

Query Processor Components:

DDL interpreter: It interprets DDL statements and records the definitions in the data dictionary.

DML compiler: It translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.

Query evaluation engine: It executes low-level instructions generated by the DML compiler.

Application Architectures:

Most users of a database system today are not present at the site of the database system, but connect to it through a network. We can therefore differentiate between client machines, on which remote database users' work, and server machines, on which the database system runs. Database applications are usually partitioned into two or three parts. They are:

1. Two – Tier Architecture

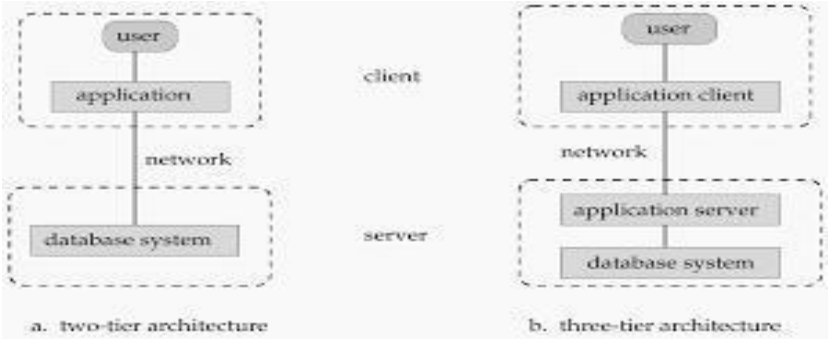
2. Three – Tier Architecture.

Two-Tier Architecture:

The application is partitioned into a component that resides at the client machine, which invokes database system functionality at the server machine through query language statements. Application program interface standards like ODBC and JDBC are used for interaction between the client and the server.

Three-Tier Architecture:

The client machine acts as merely a front end and does not contain any direct database calls. Instead, the client end communicates with an application server, usually through forms interface. The application server in turn communicates with a database system to access data. The business logic of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple clients. Three-tier applications are more appropriate for large applications, and for applications that run on the World Wide Web.



DATABASE DESIGN:

The database design process can be divided into six steps. The ER Model is most relevant to the first three steps. Next three steps are beyond the ER Model.

1. Requirements Analysis:

The very first step in designing a database application is to understand what data is to be stored in the database, what applications must be built on top of it, and what operations are most frequent and subject to performance requirements. The database designers collect information of the organization and analyzer, the information to identify the user's requirements. The database designers must find out what the users want from the database.

2. Conceptual Database Design:

Once the information is gathered in the requirements analysis step a conceptual database design is developed and is used to develop a high level description of the data to be stored in the database, along with the constraints that are known to hold over this data. This step is often carried out using the ER model, or a similar high-level data model.

3. Logical Database Design:

In this step convert the conceptual database design into a database schema (Logical Database Design) in the data model of the chosen DBMS. We will only consider **relational DBMSs**, and therefore, the task in the

logical design step is to convert an ER schema into a relational database schema. The result is a conceptual schema, sometimes called the **logical schema**, in the relational data model.

Beyond the ER Design:

The first three steps are more relevant to the ER Model. Once the logical scheme is defined designer consider the physical level implementation and finally provide certain security measures. The remaining three steps of database design are briefly described below:

4. Schema Refinement:

The fourth step in database design is to analyze the collection of relations in our relational database schema to identify potential problems, and to refine it. In contrast to the requirements analysis and conceptual design steps, which are essentially subjective, schema refinement can be guided by some elegant and powerful theory.

5. **Physical Database Design:**

In this step we must consider typical expected workloads that our database must support and further refine the database design to ensure that it meets desired performance criteria. This step may simply involve building indexes on some tables and clustering some tables, or it may involve a substantial redesign of parts of the database schema obtained from the earlier design steps.

6. Security Design:

The last step of database design is to include security features. This is required to avoid unauthorized access to database practice after all the six steps. We required Tuning step in which all the steps are interleaved and repeated until the design is satisfactory.

DBMS FUNCTIONS:

- DBMS performs several important functions that guarantee the integrity and consistency of the data in the database.
- Those functions transparent to end users and can be accessed only through the use of DBMS. They include:
    - Data Dictionary Management
    - Data Storage Management
    - Data transformation and Presentation
    - Security Management
    - Multiple Access Control
    - Backup and Recovery Management
    - Data Integrity Management
    - Database Access Languages
    - Databases Communication Interfaces

Data Dictionary Management:

- DBMS stores definitions of database elements and their relationship (Metadata) in the data dictionary.
- The DBMS uses the data dictionary to look up the required data component structures and relationships.
- Any change made in database structure is automatically recorded in the data dictionary.

Data Storage Management:

- Modern DBMS provides storage not only for data but also for related data entities.
- Data Storage Management is also important for database “performance tuning”.
- Performance tuning related to activities that make database more efficiently.

Data Transformation and Presentation:

- DBMS transforms entered data to confirm to required data structures.

- DBMS formats the physically retrieved data to make it confirms to user's logical expectations.
- DBMS also presents the data in the user's expected format.

Security Management:

- DBMS creates a security system that enforces the user security and data privacy.
- Security rules determines which users can access the database, which data items each user can access etc.

- DBA and authenticated user logged to DBMS through username and password or through Biometric authentication such as Finger print and face reorganization etc.

Multiuser Access Control:

- To provide data integrity and data consistency, DBMS uses sophisticated algorithms to ensure that multiple users can access the database concurrently without compromising the integrity of database.

Backup and Recovery Management:

- DBMS provides backup and recovery to ensure data safety and integrity.

- Recovery management deals with the recovery of database after failure such as bad sector in the disk or power failure. Such capability is critical to preserve database integrity.

Data Integrity Management:

- DBMS provides and enforces integrity rules, thus minimizing data redundancy and maximizing data consistency.

- Ensuring data integrity is especially important in transaction- oriented database systems.

Database Access Languages:

- DBMS provides data access through a query language.
- A query language is a non-procedural language i.e. it lets the user specify what must be done without specifying how it is to be done.
- SQL is the default query language for data access.

Databases Communication Interfaces:

- Current DBMS's are accepting end-user requests via different network environments.
- For example, DBMS might provide access to database via Internet through the use of web browsers such as Mozilla Firefox or Microsoft Internet Explorer.

What is Schema?

A database schema is the skeleton structure that represents the logical view of the entire database. (or)

The logical structure of the database is called as Database Schema. (or)

The overall design of the database is the database schema.

- It defines how the data is organized and how the relations among them are associated.
- It formulates all the constraints that are to be applied on the data.

EG:

STUDENT

SID	SNAME	PHNO

What is Instance?

The actual content of the database at a particular point in time. (Or)

The data stored in the database at any given time is an instance of the database

Student

Sid	Name	phno
1201	Venkat	9014901442
1202	Teja	9014774422

In the above table 1201, 1202, Venkat etc are said to be instance of student table.

UNIT-2

UNIT-2

Relational Algebra

Preliminaries

A query language is a language in which user requests to retrieve some information from the database. The query languages are considered as higher level languages than programming languages. Query languages are of two types,

Procedural Language
Non-Procedural Language

1. In procedural language, the user has to describe the specific procedure to retrieve the information from the database.

Example: The Relational Algebra is a procedural language.

2. In non-procedural language, the user retrieves the information from the database without describing the specific procedure to retrieve it.

Example: The Tuple Relational Calculus and the Domain Relational Calculus are non-procedural languages.

Relational Algebra

The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations (tables) as input and produce a new relation, on the request of the user to retrieve the specific information, as the output.

The relational algebra contains the following operations,

1) Selection 2) Projection 3) Union 4) Rename

5) Set-Difference 6) Cartesian product 7) Intersection 8) Join

9) Divide 10) Assignment

The Selection, Projection and Rename operations are called unary operations because they operate only on one relation. The other operations operate on pairs of relations and are therefore called binary operations.

1) The Selection (σ) operation:

The Selection is a relational algebra operation that uses a condition to select rows from a relation. A new relation (output) is created from another existing relation by selecting only rows requested by the user that satisfy a specified condition. The lower greek letter 'sigma σ' is used to denote selection operation.

General Syntax: **Selection condition (relation_name)**

Example: Find the customer details who are living in Hyderabad city from customer relation.

$$\sigma_{\text{city = 'Hyderabad'}} \text{ (customer)}$$

The selection operation uses the column names in specifying the selection condition. Selection conditions are same as the conditions used in the 'if' statement of any programming languages, selection condition uses the relational operators < > <= >= != . It is possible to combine several conditions into a large condition using the logical connectives 'and' represented by '∧' and 'or' represented by '∨'.

Example:

Find the customer details who are living in Hyderabad city and whose customer_id is greater than 1000 in Customer relation.

$$\sigma_{\text{city = 'Hyderabad'} \wedge \text{ customer\_id > 1000}} \text{ (customer)}$$

2) The Projection (π) operation:

The projection is a relational algebra operation that creates a new relation by deleting columns from an existing relation i.e., a new relation (output) is created from another existing relation by selecting only those columns requested by the user from projection and is denoted by letter pi (π).

The Selection operation eliminates unwanted rows whereas the projection operation eliminates unwanted columns. The projection operation extracts specified columns from a table.

Example: Find the customer names (not all customer details) who are living in Hyderabad city from customer relation.

$$\pi_{\text{customer\_name}} (\sigma_{\text{city = 'Hyderabad'}} (\textbf{customer}))$$

In the above example, the selection operation is performed first. Next, the projection of the resulting relation on the customer_name column is carried out. Thus, instead of all customer details of customers living in Hyderabad city, we can display only the customer names of customers living in Hyderabad city.

The above example is also known as relational algebra expression because we are combining two or more relational algebra operations (ie., selection and projection) into one at the same time.

Example: Find the customer names (not all customer details) from customer relation.

$$\pi_{\text{customer\_name}} (\textbf{customer})$$

The above stated query lists all customer names in the customer relation and this is not called as relational algebra expression because it is performing only one relational algebra operation.

3) The Set Operations: (Union, Intersection, Set-Difference, Cartesian product **)**

i) Union ' ∪ ' Operation:

The union denoted by '∪'. It is a relational algebra operation that creates a union or combination of two relations. The result of this operation, denoted by d ∪ b is a relation that includes all tuples that all either in d or in b or in both d and b, where duplicate tuples are eliminated.

Example: Find the customer_id of all customers in the bank who have either an account or a loan or both.

$$\pi_{\text{customer\_id}} (\textbf{depositor}) \cup \pi_{\text{customer\_id}} (\textbf{borrower})$$

To solve the above query, first find the customers with an account in the bank. That is $\pi_{\text{customer\_id}}$ **(depositor)**. Then, we have to find all customers with a loan in the bank, $\pi_{\text{customer\_id}}$ **(borrower)**. Now, to answer the above query, we need the union of these two sets, that is, all customer names that appear in either or

both of the two relations by $\pi_{customer\_id}$ (depositor) $\cup$ $\pi_{customer\_id}$ (borrower)

If some customers A, B and C are both depositors as well as borrowers, then in the resulting relation, their customer ids will occur only once because duplicate values are eliminated.

Therefore, for a union operation d $\cup$ b to be valid, we require that two conditions to be satisfied,

i) The relations depositor and borrower must have same number of attributes / columns.

ii) The domains of i^{th} attribute of depositor relation and the i^{th} attribute of borrower relation must be the same, for all i.

- **The Intersection ' $\cap$ ' Operation:**

The intersection operation denoted by ' $\cap$'. It is a relational algebra operation that finds tuples that are in both relations. The result of this operation, denoted by d $\cap$ b, is a relation that includes all tuples common in both depositor and borrower relations.

Example: Find the customer_id of all customers in the bank who have both an account and a loan.

$$\pi_{customer\_id}\textbf{(depositor)} \cap \pi_{customer\_id}\textbf{(borrower)}$$

The resulting relation of this query, lists all common customer ids of customers who have both an account and a loan. Therefore, for an intersection operation d $\cap$ b to be valid, it requires that two conditions to be satisfied as was the case of union operation stated above.

iii) The Set-Difference ' – ' Operation:

The set-difference operation denoted by' –'. It is a relational algebra operation that finds tuples that are in one relation but are not in another.

Find the customer id of all customers in the bank who have an account but not a loan

Example:

$$\pi_{customer\_id}(\textbf{depositor}) - \pi_{customer\_id}(\textbf{borrower})$$

The resulting relation for this query, lists the customer ids of all customers who have an account but not a loan. Therefore a difference operation d – b to be valid, it requires that two conditions to be satisfied as was case of union operation stated ablove.

iv) The Cross-product (or) Cartesian Product ‘ X ’ Operation:

The Cartesian-product operation denoted by a cross ‘X’. It is a relational algebra operation which allows to combine information from who relations into one relation.

Assume that there are n_1 tuple in borrower relation and n_2 tuples in loan relation. Then, the result of this operation, denoted by r = borrower X loan, is a relation ‘r’ that includes all the tuples formed by each possible pair of tuples one from the borrower relation and one from the loan relation. Thus, ‘r’ is a large relation containing $n_1 * n_2$ tuples.

The drawback of the Cartesian-product is that same attribute name will repeat.

Example: Find the customer_id of all customers in the bank who have loan > 10,000.

$$\pi_{customer\_id}(\sigma_{borrower.loan\_no=loan.loan\_no}((\sigma_{borrower.loan\_no=}(\textbf{borrower X loan}))))$$

That is, get customer_id from borrower relation and loan_amount from loan relation. First, find Cartesian product of borrower X loan, so that the new relation contains both customer_id, loan_amoount with each combination. Now, select the amount, by $\sigma_{bloan\_ampunt} > 10000$.

So, if any customer have taken the loan, then borrower.loan_no = loan.loan_no should be selected as their entries of loan_no matches in both relation.

4) The Renaming " ρ " Operation:

The Rename operation is denoted by rho 'ρ'. It is a relational algebra operation which is used to give the new names to the relation algebra expression. Thus, we can apply the rename operation to a relation 'borrower' to get the same relation under a new name. Given a relation 'customer', then the expression returns the same relation 'customer' under a new name 'x'.

$$\rho_x (\textbf{customer})$$

After performed this operation, Now there are two relations, one with customer name and second with

'x' name. The 'rename' operation is useful when we want to compare the values among same column attribute in a relation.

Example: Find the largest account balance in the bank.

$$\pi_{\text{account.balance}} (\sigma_{\text{account.balance} > \text{d.balance}} (\text{account} \times \rho_d (\text{account})))$$

If we want to find the largest account balance in the bank, Then we have to compare the values among same column (balance) with each other in a same relation account, which is not possible.

So, we rename the relation with a new name'd'. Now, we have two relations of account, one with account name and second with 'd' name. Now we can compare the balance attribute values with each other in separate relations.

5) The Joins " ⋈ " Operation:

The join operation, denoted by join ' ⋈ '. It is a relational algebra operation, which is used to combine

(join) two relations like Cartesian-product but finally removes duplicate attributes and makes the operations (selection, projection, ..) very simple. In simple words, we can say that join connects relations on columns containing comparable information.

There are three types of joins,

i) Natural Join

ii) Outer Join

iii) Theta Join (or) Conditional Join

i) Natural Join:

The natural join is a binary operation that allows us to combine two different relations into one relation and makes the same column in two different relations into only one-column in the resulting relation. Suppose we have relations with following schemas, which contain data on full-time employees.

employee (emp_name, street, city) and

employee_works(emp_name, branch_name, salary)

The relations are,

employee relation

emp_name	street	city
Coyote	Town	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Vally
Williams	Seaview	Seattle

emp_name	branch_name	salary
Coyote	Mesa	15000
Rabbit	Mesa	12000
Gates	Redmond	25000
Williams	Redmond	23000

employee_works relation

If we want to generate a single relation with all the information (emp_name, street, city, branch_name and salary) about full-time employees. then, a possible approach would be to use the natural-join operation as follows,

employee ⋈ employee_works

The result of this expression is the relation,

emp_name	street	city	branch_name	salary
Coyote	Town	Hollywood	Mesa	15000
Rabbit	Tunnel	Carrotville	Mesa	12000
Williams	Seaview	Seattle	Redmond	23000

result of Natural join

We have lost street and city information about Smith, since tuples describing smith is absent in employee_works. Similarly, we have lost branch_name and salary information about Gates, since the tuple

describing Gates is absent from the employee relation. Now, we can easily perform select or reject query on new join relation.

Example: Find the employee names and city who have salary details.

$\pi_{emp\_name, salary, city}$ **(employee ⋈ employee_works)**

The join operation selects all employees with salary details, from where we can easily project the employee names, cities and salaries. Natural Join operation results in some loss of information.

ii) Outer Join:

The drawback of natural join operation is some loss of information. To overcome the drawback of natural join, we use outer-join operation. The outer-join operation is of three types,

a) Left outer-join ( )
b) Right outer-join ( )
c) Full outer-join ( )

a) Left Outer-join:

The left outer-join takes all tuples in left relation that did not match with any tuples in right relation, adds the tuples with null values for all other columns from right relation and adds them to the result of natural join as follows,

The relations are,

emp_name	street	city
Coyote	Town	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Valley
Williams	Seaview	Seattle

employee relation

emp_name	branch_name	salary
Coyote	Mesa	15000
Rabbit	Mesa	12000
Gates	Redmond	25000
Williams	Redmond	23000

employee_works relation

The result of this expression is the relation,

emp_name	street	City	branch_name	salary
Coyote	Town	Hollywood	Mesa	15000
Rabbit	Tunnel	Carrotville	Mesa	12000
Smith	Revolver	Valley	Null	null
Williams	Seaview	Seattle	Redmond	23000

result of Left Outer-join

b) Right Outer-join:

The right outer-join takes all tuples in right relation that did not match with any tuples in left relation, adds the tuples with null values for all other columns from left relation and adds them to the result of natural join as follows,

The relations

emp_name	street	city
Coyote	Town	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Valley
Williams	Seaview	Seattle

emp_name	branch_name	Salary
Coyote	Mesa	15000
Rabbit	Mesa	12000
Gates	Redmond	25000
Williams	Redmond	23000

The result of this expression is the relation,

emp_name	street	City	branch_name	salary
Coyote	Town	Hollywood	Mesa	15000
Rabbit	Tunnel	Carrotville	Mesa	12000
Gates	null	Null	Redmond	25000
Williams	Seaview	Seattle	Redmond	23000

result of Right Outer-join

c) Full Outer-join:

The full outer-join operation does both of those operations, by adding tuples from left relation that did not match any tuples from the reight relations, as well as adds tuples from the right relation that did not match any tuple from the left relation and adding them to the result of natural join as follows,

The relations are,

emp_name	street	city
Coyote	Town	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Valley
Williams	Seaview	Seattle

employee relation

emp_name	branch_name	salary
Coyote	Mesa	15000
Rabbit	Mesa	12000
Gates	Redmond	25000
Williams	Redmond	23000

employee_works relation

The result of this expression is the relation,

emp_name	street	City	branch_name	salary
Coyote	Town	Hollywood	Mesa	15000
Rabbit	Tunnel	Carrotville	Mesa	12000

Smith	Revolver	Valley	Null	null
Gates	null	Null	Redmond	25000
Williams	Seaview	Seattle	Redmond	23000

result of Full Outer-join

iii) Theta Join (or) Condition join:

The theta join operation, denoted by symbol " $\bowtie_\theta$ " . It is an extension to the natural join operation that combines two relations into one relation with a selection condition (θ).

The theta join operation is expressed as employee $\bowtie_{salary < 19000}$ employee_works and the resulting is as follows,

employee $\bowtie_{salary > 20000}$ employee_works

There are two tuples selected because their salary greater than 20000 (salary > 20000). The result of theta join as follows,

The relations are,

emp_name	street	city
Coyote	Town	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Valley
Williams	Seaview	Seattle

emp_name	branch_name	salary
Coyote	Mesa	15000
Rabbit	Mesa	12000
Gates	Redmond	25000
Williams	Redmond	23000

The result of this expression is the relation,

emp_name	street	City	branch_name	salary
Gates	null	Null	Redmond	25000
Williams	Seaview	Seattle	Redmond	23000

result of Theta Join (or) Condition Join

6) The Division " ÷ " Operation:

The division operation, denoted by **" ÷ "**, is a relational algebra operation that creates a new relation by selecting the rows in one relation that does not match rows in another relation.

Let, Relation A is (x_1, x_2, …., x_n, y_1, y_2, …, y_m) and
Relation B is (y_1, y_2, …, y_m),

Where, y_1, y_2, …, y_m tuples are common to the both relations A and B with same domain
compulsory.

Then, A ÷ B = new relation with x_1, x_2, …., x_n tuples. Relation A and B represents the dividend and devisor respectively. A tuple 't' is in a ÷ b, if and only if two conditions are to be satisfied,

- t is in π_{A-B} (r)
- for every tuple t_b in B, there is a tuple t_a in A satisfying the following two things,

1. $t_a[B] = t_b[B]$
2. $t_a[A-B] = t$

Relational Calculus

Relational calculus is an alternative to relational algebra. In contrast to the algebra, which is procedural, the relational calculus is non-procedural or declarative.

It allows user to describe the set of answers without showing procedure about how they should be computed. Relational calculus has a big influence on the design of commercial query languages such as SQL and QBE (Query-by Example).

Relational calculus are of two types,

- Tuple Relational Calculus (TRC)

- Domain Relational Calculus (DRC)

Variables in TRC takes tuples (rows) as values and TRC had strong influence on SQL.

Variables in DRC takes fields (attributes) as values and DRC had strong influence on QBE.

i) Tuple Relational Calculus (TRC):

The tuple relational calculus, is a non-procedural query language because it gives the desired
information without showing procedure about how they should be computed.

A query in Tuple Relational Calculus (TRC) is expressed as **{ T | p(T) }**

Where, T - tuple variable,
P(T) - 'p' is a condition or formula that is true for 't'.

In addition to that we use,

T[A] - to denote the value of tuple t on attribute A and
$T \in r$ - to denote that tuple t is in relation r.

Examples:

1) Find all loan details in loan relation.

{ t | t ∈ loan }

This query gives all loan details such as loan_no, loan_date, loan_amt for all loan table in a bank.

2) Find all loan details for loan amount over 100000 in loan relation.

{ t | t ∈ loan ∧ t [loan_amt] > 100000 }

This query gives all loan details such as loan_no, loan_date, loan_amt for all loan over 100000 in a loan table in a bank.

ii) Domain Relational Calculus (DRC):

A Duple Relational Calculus (DRC) is a variable that comes in the range of the values of domain (data types) of some columns (attributes).

A Domain Relational Calculus query has the form,

{ < x_1, x_2, …., x_n > | p(< x_1, x_2, …., x_n >) }

Where, each x_i is either a domain variable or a constant and p(< x_1, x_2, …., x_n >) denotes a DRC formula.

A DRC formula is defined in a manner that is very similar to the definition of a TRC formula. The main difference is that the variables are domain variables.

Examples:

1) Find all loan details in loan relation.

{ < N, D, A > | < N, D, A > ∈ loan }

This query gives all loan details such as loan_no, loan_date, loan_amt for all loan table in a bank. Each column is represented with an initials such as N- loan_no, D – loan_date, A – loan_amt. The condition < N, D, A > ∈ loan ensures that the domain variables N, D, A are restricted to the column domain.

2.3.1 Expressive power of Algebra and Calculus

The tuple relational calculus restricts to safe expressions and is equal in expressive power to relational algebra. Thus, for every relational algebra expression, there is an equivalent expression in the tuple relational calculus and for tuple relational calculus expression, there is an equivalent relational algebra expression.

A safe TRC formula Q is a formula such that,

- For any given I, the set of answers for Q contains only values that are in dom(Q, I).

- For each sub expression of the form ∃R(p(R)) in Q, if a tuple r makes the formula true, then r contains only constraints in dom(Q, I).

3) For each sub expression of the form ∀R(p(R)) in Q, if a tuple r contains a constant that is not in

dom(Q, I), then r must make the formula true.

The expressive power of relational algebra is often used as a metric how powerful a relational database query language is. If a query language can express all the queries that we can express in relational algebra, it is said to be relationally complete. A practical query language is expected to be relationally complete. In addition, commercial query languages typically support features that allow us to express some queries that cannot be expressed in relational algebra.

When the domain relational calculus is restricted to safe expression, it is equivalent in expressive power to the tuple relational calculus restricted to safe expressions. All three of the following are equivalent,

- The relational algebra
- The tuple relational calculus restricted to safe expression
- The domain relational calculus restricted to safe expression

TRIGGERS IN SQL:

Trigger:

Triggers are stored programs, which are automatically executed or fired when some events occur

A Trigger can be defined as a program that is executed by DBMS whenever updations are specified on database tables.

It is like an event which occurs whenever a change is done to the tables or columns of tables.

Only DBA can specify the triggers.

Triggers are, in fact, written to be executed in response to any of the following events:

A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).

Database definition (DDL) statements (CREATE, ALTER, or DROP).

A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers:

Triggers can be written for the following purposes:

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

General form of a Trigger:

The general form of the trigger includes the following:

Event
Condition
Action

Event:

Event describes the modifications done on the database which lead to the activation of trigger.
The following comes under the category of events:

1). Inserting, updating, deleting columns of the tables or rows of the tables may activate trigger.
2). Creating, altering or dropping any database object may also lead to activation of triggers.
3). An error message occur or user log-on or log-off may also activate the trigger.

Condition:

Conditions are used to specify the actions to be taken when the corresponding event occurs and the condition evaluates to true.
If the condition evaluated to true then the respective action to be taken otherwise action is rejected.

Action:

Action specifies the action to be taken place when the corresponding event occurs and the condition evaluates to true.
An action is a collection of SQL statements that are executed as a part of trigger activation.

Types of Triggers:

Triggers are classified based on levels. They are:

There are two levels of triggers to be invoked. They are:

1. Statement level triggers

2. Row level triggers

Statement Level Triggers:

These triggers executed only once for multiple rows which are affected by trigger action.

Row-level Triggers:

These triggers run for each row of the table that is affected by the event of triggers.

The following comes under the row-level triggers. They are:

1. **BEFORE Trigger:** BEFORE trigger execute before the triggering DML statement (INSERT, UPDATE, DELETE) execute. Triggering SQL statement is may or may not execute, depending on the BEFORE trigger conditions block.

2. **AFTER Trigger**: AFTER trigger execute after the triggering DML statement (INSERT, UPDATE, DELETE) executed. Triggering SQL statement is executed as soon as followed by the code of trigger before performing Database operation.

3. **ROW Trigger:** ROW triggers fire for each and every record which are performing INSERT, UPDATE, DELETE from the database table. If row deleting is define as trigger event, when trigger file, deletes the five rows each times from the table.

4. **Statement Trigger**: Statement trigger fire only once for each statement. If row deleting is define as trigger event, when trigger file, deletes the five rows at once from the table.

5. **Combination Trigger** : Combination trigger are combination of two trigger type,
    a. **Before Statement Trigger**: Trigger fire only once for each statement before the triggering DML statement.
    b. **Before Row Trigger**: Trigger fire for each and every record before the triggering DML statement.
    c. **After Statement Trigger:** Trigger fire only once for each statement after the triggering DML statement executing.

After Row Trigger: Trigger fire for each and every record after the triggering DML statement executing.

UNIT-3

UNIT-3

INTRODUCTION TO SCHEMA REFINEMENT:

- Data redundancy means duplication of data. It causes duplicate data at different locations which destroys the integrity of the database and wastage of storage space.
  The problems of redundancy are:
- Wasted Storage Space. 2. More difficult Database Updates. 3. A Possibility of Inconsistent data.

FUNCTIONAL DEPENDENCY:

- The normalization theory based on the fundamental notion of FUNCTIONAL DEPENDENCY.
- Given a relation R, attribute A is functionally dependent on attribute B if each value of value of A in R is associated with precisely one value of B.

  (OR)

  In other words, attribute A is functionally dependent on B if and only if, for each value of B, there is exactly one value of A.

  (OR)

- Functional dependency is a relationship that exists when one attribute uniquely determines another attribute.
- If R is a relation with attributes X and Y, a functional dependency between the attributes is represented as
  X->Y, which specifies Y is functionally dependent on X.
- Here X is a determinant set and Y is a dependent attribute. Each value of X is associated precisely with one Y value.
- Consider the following table EMPLOYEE

EMPLOYEE

CODE	NAME	CITY
E1	Mac	Delhi
E2	Sandra	CA
E3	Henry	Paris

- Given a particular value of CODE, there is exactly one corresponding value for NAME.
- For example, for CODE E1 there is exactly one value of NAME, Mac. Hence NAME is functionally dependent on code.
- Similarly, there is exactly one value of CITY for each value of CODE. Hence, the attribute CITY is functionally dependent on the attribute CODE.
- The attribute CODE is the determinant. You can also say that CODE determines CITY and NAME.

  i.e

  Code----→Name

  Code----→City

  And also Code----→Name, City

From above statements, we can conclude that,

Where code is determinant and name, City are dependent and we can say that Name, City are functionally dependent on Code.

Rules about Functional Dependencies

The set of all FD's implied by a given set F of FD's is called the closure of F, denoted as F^+. The following three rules, called armstrong's axioms, can be applied repeatedly to compute all FDs implied by a set of FDs. Here, we use X, Y and Z to denote sets of attribute over a relation schema R,

Rule 1: **Reflexivity:** if $X \supseteq Y$, then $X \rightarrow Y$.

Rule 2: **Augmentation:** if $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z.

Rule 3: **Transitivity:** if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

It is convenient to use some additional rules while reasoning about F^+.

Rule 4: **Union:** If $x \rightarrow Y$ and $X \rightarrow z$, then $X \rightarrow YZ$.

Rule 5: **Decomposition:** If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$.

Rule 6: **Pseudotranstivity:** If $X \rightarrow Y$ and $Y \rightarrow P$, then $XZ \rightarrow P$.

LOSSLESS JOIN DECOMPOSITION:

A decomposition {R1, R2,…, Rn} of a relation R is called a lossless decomposition for R if the natural join of R1, R2,…, Rn produces exactly the relation R.

R : relation

F : set of functional dependencies on R

X,Y : decomposition of R

Decomposition is lossles if :

- X ∩ Y → X, that is: all attributes common to both X and Y functionally determine ALL the attributes in X **OR**
- X ∩ Y → Y, that is: all attributes common to both X and Y functionally determine ALL the attributes in Y

In other words, if X ∩ Y forms a superkey of either X or Y, the decomposition of R is a lossless decomposition.

Example:

Given:

Lending-schema = (branch-name, branch-city, assets, customer-name, loan-number, amount)

Required FD's:

branch-name branch-city assets

loan-number amount branch-name

Decompose Lending-schema into two schemas:

Branch-schema = (branch-name, branch-city, assets)

Loan-info-schema = (branch-name, customer-name, loan-number, amount)

- Since *branch-name branch-city assets,* the augmentation rule for FD implies that:

  branch-name branch-name branch-city assets

- Since *Branch-schema ∩ Loan-info-schema* = {*branch-name*}

Thus, this decomposition is Lossless decomposition.

DEPENDENCY PRESERVATION PROPERTY:

A decomposition of a relation R into R1, R2, R3, …, Rn is dependency preserving decomposition with respect to the set of Functional Dependencies F that hold on R only if the following is hold;

(F1 U F2 U F3 U … U Fn)+ = F+

where,

F1, F2, F3, …, Fn – Sets of Functional dependencies of relations R1, R2, R3, …, Rn.

(F1 U F2 U F3 U … U Fn)+ - Closure of Union of all sets of functional dependencies.

F+ - Closure of set of functional dependency F of R.

If the closure of set of functional dependencies of individual relations R1, R2, R3, …, Rn are equal to the set of functional dependencies of the main relation R (before decomposition), then we would say the decomposition D is lossless dependency preserving decomposition.

Discussion with Example of Non-dependency preserving decomposition:

Dependency preservation is a concept that is very much related to Normalization process. Remember that the solution for converting a relation into a higher normal form is to decompose the relation into two or more relations. This is done using the set of functional dependencies identified in the lower normal form state.

For example, let us assume a relation R (A, B, C, D) with set of functional dependencies F = {A→B, B→C, C→D}. There is no partial dependency in the given set F. Hence, this relation is in 2NF.

Is R (A, B, C, D) in 3NF? No. The reason is Transitive Functional Dependency. How do we convert R into 3NF? The solution is decomposition.

Assume that we decompose R(A, B, C, D) into R1(A, C, D) and R2(B, C).

In R1(A, C, D), the FD $C \rightarrow D$ holds. In R2(B, C), the FD $B \rightarrow C$ holds. But, there is no trace of the FD $A \rightarrow B$. Hence, this decomposition does not preserve dependency.

What wrong would the above decomposition cause?

In R, the following were held;

value of B depends on value of A,

value of C depends on value of B,

value of D depends on value of C.

after decomposition, the relations R1 and R2 are holding the following;

value of C depends on value of B,

value of D depends on value of C.

The dependency $A \rightarrow B$ is missing. This causes acceptance of any values for B in R2. It causes duplicate values to be entered into R2 for B which may not depend on A. If we would like to avoid this type of duplication, then we need to perform a join operation between R1 and R2 to accept a new value for B which is costlier operation. Hence, we demand the decomposition to be a dependency preserving decomposition.

INTRODUCTION TO NORMALIZATION:

Redundancy:

- Redundancy means repetition of data.
- Redundancy increases the time involved in updating, adding, and deleting data.
- It also increases the utilization of disk space and hence, disk I/O increases.

DEFINITION OF NORMALIZATION:

- Normalization is scientific method of breaking down complex stable structures into simple table structures by using certain rules.
- Using this method, you can, reduce redundancy in a table and elimates the problems of inconsistency and disk space usage.
- We can also ensure that there is no loss of information.

Benefits of Normalization:

- Normalization has several benefits.
- It enables faster sorting and index creation, more clustered indexes, few indexes per table, few NULLs, and makes the database compact.
- Normalization helps to simplify the structure of table. The performance of an application is directly linked to database design.
- A poor design hinders the performance of the system.
- The logical design of the database plays the foundation of an optical database.
- The following are the some rules that should be followed to achieve a good data base design .They are:
    - Each table should have an identifier.
    - Each table should store data for single type entity.
    - Columns that accept NULLs should be avoided.
    - The repetition of values or columns should be avoided.

NORMAL FORMS:

- The normalization results in the formation of that satisfy certain specified rules and represent certain normal forms.
- The normal forms are used to ensure that several of anomalies and inconsistencies are not introduced in the database.

- A table structure is always in a certain normal form. Several normal forms have been identified.
- The following are the most important and widely used normal forms are:
    - First normal form(1NF)
    - Second normal form(2NF)
    - Third normal form(3NF)
    - Boyce-Codd Normal Form (BCNF)
    - Fourth Normal Form (4NF)

- The first, second and third normal forms are originally defined by Dr. E. F. Codd.
- Later, Boyce and Codd introduced another form called the Boyce-Codd Normal form.

FIRST NORMAL FORM (1NF):

- .A table is said to be 1NF when each cell of the table contains precisely one value.

Consider the following table PROJECT.

PROJECT:

ECODE	DEPT	DEPTHEAD	PROJCODE	HOURS
E101	Systems	E901	P27 P51 P20	90 101 60
E305	Sales	E906	P27 P22	109 98
E508	Admin	E908	P51 P27	NULL 72

- The data in the table is not normalized because the cells in the PROJCODE and HOURS have more than one value.
- By applying the INF to the PROJECT table, the resultant table is as follows:

PROJECT:

ECODE	DEPT	DEPTHEAD	PROJCODE	HOURS
E101	Systems	E901	P27	90
E101	Systems	E901	P51	101
E101	Systems	E901	P20	60
E305	Sales	E906	P27	109
E305	Sales	E906	P22	98
E508	Admin	E908	P51	NULL

SECOND NORMAL FORM (2NF)

- A table is said to be in 2NF when it is in 1NF and every attribute in the row is functionally dependent upon the whole key, and not just part of the key.
- Consider the PROJECT table.

PROJECT

ECODE
PROJCODE
DEPT
DEPTHEAD
HOURS

The table has following rows.

ECODE	PROJCODE	DEPT	DEPTHEAD	HOURS
E101	P27	Systems	E901	90
E305	P27	Finance	E909	10
E508	P51	Admin	E908	NULL
E101	P51	Systems	E901	101

E101	P20	Systems	E901	60
E508	P27	Admin	E908	72

- The above table satisfies the definition of 1NF and now we have to now check if it satisfies 2NF.
- In the above table, for each value of ECODE, there is more than one value of HOURS.
- For example, for ECODE, E101, there are 3 values of HOURS-90, 101 and 60. Hence, HOURS is not functionally dependent on ECODE.
- Similarly, for each value of PROJCODE, there is more than 1 value of HOURS.
- For example, for PROJCODE P27, there are three values of HOURS-90, 10 and 72.
- However, for a combination of the ECODE and PROJCODE values, there is exactly one value of HOURS. Hence, HOURS is functionally dependent on the whole key, ECODE+PROJCODE.
- Therefore DEPT is functionally dependent on the part of the key (Which is ECODE) and not functionally dependent on the whole key (ECODE+PROJCODE).
- For the table to be in 2NF, the non-key attributes must be functionally dependent on the whole key and not the part of the key.

GUIDELINES FOR CONVERTING THE TABLE TO 2NF:

- Find and remove attributes that are functionally dependent on only a part of the key and not on the whole key. Place them in a different table.
- Group the remaining attributes.

After applying the 2NF:

- After applying the 2Nf to above table, the resultant tables are as follows:

EMPLOYEE -DEPT

ECODE	DEPT	DEPTHEAD
E101	Systems	E901
E305	Finance	E909
E508	Admin	E908
E508	Admin	E908

PROJECT

ECODE	PROJCODE	HOURS
E101	P27	90
E101	P51	101
E101	P20	60
E305	P27	10
E508	P51	NULL
E508	P27	72

THIRD NORMAL FORM (3NF):

A table is said to be in 3NF when it is in 2NFand every non-key attribute is functionally dependent only on the primary key.

(OR)

A relation is said to be in Third Normal Form, if a table must be follow the following. They are

- The table must be in 2 NF prior.
- No non-prime attribute is transitively dependent on prime key attribute.

Consider the table EMPLOYEE.

ECODE	DEPT	DEPTHEAD
E101	Systems	E901
E305	Finance	E909
E402	Sales	E906
E508	Admin	E908
E607	Finance	E909
E608	Finance	E909

- The above table is in 1NF because each row of a table contains atomic value.

- The primary key is in the EMPLOYEE table is ECODE, for each value of ECODE, there is exactly one value of DEPT. Hence, the attribute DEPT is functionally dependent on the primary key, ECODE.
- Similarly, for each value of ECODE, there is exactly one value of DEPTHEAD. Therefore, DEPTHEAD is functionally dependent on the primary key ECODE.
- Hence, all the attributes are functionally dependent on the whole key, ECODE. Hence the table is in 2NF.
- However, the attribute DPETHEAD is dependent on the attribute DEPT also. As per 3NF, all non-key attributes have to be functionally dependent only on the primary key.
- This table is not in 3NF since DEPTHEAD is functionally dependent on DEPT, Which is not a primary key.

GUIDELINES FOR CONVERTING A TABLE TO 3NF

  - Find and remove non-key attributes that are functionally dependent on attributes that are not primary key. Place them in a different table.
  - Group the remaining attributes.

- To convert the table employee into 3NF, we must remove the column DEPTHEAD since it is not functionally dependent on only the primary key ECODE, and place it in another table called DEPARTMENT along with the attribute DEPT that is functionally dependent on ECODE

After applying the 3NF:

EMPLOYEE

ECODE	DEPT
E101	Systems
E305	Finance
E402	Sales
E508	Admin
E607	Finance
E608	Finance

ECODE	DEPTHEAD
Systems	E901
Sales	E906
Admin	E908
Finance	E909

- Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form
- A relation is in the BCNF if and only if every determinant is a candidate key.

Consider the following PROJECT table.

PROJECT:

ECODE	NAME	PROJCODE	HOURS
E1	Veronica	P2	48
E2	Anthony	P5	100
E3	Mac	P6	15
E4	Susan	P2	250
E4	Susan	P5	75
E1	Veronica	P5	40

- This table has redundancy. If the name of the employee is modified, the change will have to be made consistent across the table, Otherwise there will be inconsistencies.
- The following are candidate keys for above table:

  ECODE+PROJCODE

  NAME+PROJCODE

  - **HOURS** is functionally dependent on **ECODE+PROJCODE**.
  - **HOURS** is also functionally dependent on **NAME+PROJCODE.**
  - **NAME** is functionally dependent on **ECODE.**
  - **ECODE** is functionally dependent on **NAME.**
- Multiple candidate keys that is ECODE + PROJCODE and NAME+ PROJCODE.
- The candidate keys are composite.
- The candidate keys overlap since the attribute PROJCODE is common.

This is a situation that requires conversion to BCNF. The table is essentially in the third NF. The only non-key item is HOURS, which is dependent, on the whole key, that is, ECODE+PROJCODE or PROJCODE.

GUIDELINES FOR CONVERTING A TABLE TO BCNF:

- Find and remove that overlapping candidate keys. Place the part of the candidate key and the attribute it is functionally dependent on, in a different table.

- Group the remaining items into a table.

After applying the BCNF

EMPLOYEE

ECODE	NAME
E1	Veronica
E2	Anthony
E3	Mac
E4	Susan

PROJECT

ECODE	PROJCODE	HOURS
E1	P2	48
E2	P5	100
E3	P6	15
E4	P2	250
E4	P5	75
E1	P5	40

Fourth Normal Form (4NF):

- If the database table contains multiple multi valued attributes then that table is said to be in 4 NF.
- For example, consider the the possibility that an employee can have multiple assignments and also have multiple service organizations.

VOLUNTEER-1

ENO	ORG_CODE	ASSIGN_NUM
10123	RC	1
10123	UW	3
10123		4

VOLUNTEER-2

ENO	ORG_CODE	ASSIGN_NUM
10123	RC	

10123	UW	
10123		**1**
10123		**2**

- In the above table, employee 10123 does volunteer work for Red Cross (RC) and United Way (UW).
- I addition that, the same employee might be assigned to work on three projects such as 1, 2 and 4.
- The attributes ORG_CODE and ASSIGN_NUM each may have many different attributes.i.e the table contain two sets of Multivalued attributes.
- Your tables conform to the following two rules
    1. All attributes must be dependent on the primary key.
    2. No row may contain two or more multi valued facts about entity

MULTIVALUED DEPENDENCIES (MVD)

Given a relation schema R, let X and Y be subsets of attributes of R (X and Y need not be distinct). Then the multivalued dependency denoted as $X \rightarrow\rightarrow Y$ satisfies in a relation R, if given two tuples t_1 and t_2 in R with $t_1(X) = t_2(X)$, where $R(t_1, t_2, t_3, t_4)$ have the same X value i.e.,

$$t_1(X) = t_2(X) = t_3(X) = t_4(X)$$

Whereas the Y values of t_1 and t_3 are same and the Y values of t_2 and t_4 are same, i.e., $t_1(Y) = t_3(Y) = t_2(Y) = t_4(Y)$.

The R – X – Y value of t_1 and t_4 are same and the R – X – Y values of t_2 and t_3 are same i.e.,

$$t_1(R – X – Y) = t_4(R – X – Y)$$

$$t_2(R – X – Y) = t_3(R – X – Y$$

X	Y	Z	tuples
a	b_1	c_1	- tuple t_1
a	b_2	c_2	- tuple t_2
a	b_1	c_2	- tuple t_3
a	b_2	c_1	- tuple t_4

An illustration of MVD definition

Therefore, for each X value in such a relation, there will be a set of Y values associated with it. This association between the X and Y values does not depend on the values of other attributes in the relation.

In the above example, two tuples t_1, t_2 in relation R defined on relation schema R with the same X value (i.e., 'a'). Exchange the Y values (i.e., b_1 & b_2) of these tuples t_1, t_2 and obtained the Y values of tuples t_3, t_4. (i.e., b_1 & b_2) (that is, same values as t_1, t_2). then tuples t_3 and t_4 must also be in R.

Thus, the above example is illustrated as MVD X → Y holds over R.

The first three rules are Armstrong's Axioms,

Rule 1: **Reflexivity:** if X Y, then X → Y

Rule 2: **Augmentation:** if X → Y, then XZ → YZ for any Z.

Rule 3: **Transitivity:** if X → Y and Y → Z, then X → Z.

Three of the additional rules involve only in MVDs,

Rule 4: **MVD Complementation:** If X →→ Y, then X →→ R – XY.

Rule 5: **MVD Augmentation:** If X →→ Y, then W □Z, then WX →→ YZ.

Rule 6: **MVD Transitivity:** if X →→ Y and Y →→ Z, then X →→ (Z – Y).

The following rules are related to both FDs and MVDs.

Rule 7: **Replication:** If X →→Y, then X →→Y.

Rule 8: **Coalescence:** If X →→Y and there is a W such that W ⊔⊔Y is empty. W → Z and Y ⊔⊔Z then X → Z.

Observe the above replication rule states that every FD is also a MVD.

5TH NORMAL FORM :

Definition 1 :
A relation R is in 5NF if and only if every join dependency in R is implied by the candidate keys of R.

Definition 2 :
A relation decomposed into two relations must have **loss-less join Property**, which ensures that no spurious or extra tuples are generated, when relations are reunited through a natural join.

What is a Join Dependency(JD) ??

Let ***R*** be a relation. Let ***A, B, …, Z*** be arbitrary subsets of ***R's*** attributes. ***R*** satisfies the ***JD * (A, B, …, Z)***
if and only if ***R*** is equal to the join of its projections on ***A, B, …, Z.***

A join dependency ***JD(R1, R2, …, Rn)*** specified on relation schema ***R,*** is a trivial JD, if one of the relation schemas ***Ri*** in ***JD(R1, R2, ….,Rn)*** is equal to R.

Join dependency is used in the following case :

When there is no lossless join decomposition of ***R*** into two relation schemas, but there is a lossless join decompositions of ***R*** into more than two relation schemas.

Point : A join dependency is very difficult in a database, hence normally not used.

Example :

Consider a relation ***ACP(Agent, Company, Product)***

ACP :				Meaning of the tuples
Agent(A)	Company(C)	Product(P)	⇒	Agent sells Company's Products.

A1	PQR	Nut	⇒	A1 sells PQR's Nuts and Screw.
A1	PQR	Screw		
A1	XYZ	Bolt	⇒	A1 sells XYZ's Bolts.
A2	PQR	Bolt	⇒	A2 sells PQR's Bolts.

The table is in 4 NF as it does not contain multivalued dependency. But the relation**contains redundancy** as *A1 is an agent for PQR twice*. But there is no way of eliminating this redundancy without losing information.

Suppose that the table is decomposed into its two relations, ***R1*** and ***R2***.

The redundancy has been eliminated by decomposing ***ACP*** relation, but the information about which companies make which products and which agents supply which product has been lost. The natural join of these relations over the 'agent' columns is:

R_{12} :

Agent	Company	Product
A1	PQR	Nut
A1	PQR	Screw
A1	PQR	Bolt

A1	XYZ	Nut
A1	XYZ	Screw
A1	XYZ	Bolt
A2	PQR	Bolt

Hence, the decomposition of ***ACP* is a *lossy join decomposition*** as the natural join table is spurious, since it contains extra tuples(shaded) that gives incorrect information.

But now, suppose the original relation ***ACP*** is decomposed into 3 relations :

- **R1(Agent, Company)**
- **R2(Agent, Product)**
- **R3(Company, Product)**

The result of the natural join of ***R1*** and ***R2*** over 'Agent' (already Calculated ***R12***) and then, natural join of ***R12*** and ***R3*** over 'Company' & 'Product' is –

R_{123} :

Agent	**Company**	**Product**
A1	PQR	Nut
A1	PQR	Screw

A1	PQR	Bolt
A1	XYZ	Bolt
A2	PQR	Bolt

Again, we get an extra tuple shown as by shaded portion.

Hence, it has to be accepted that it is not possible to eliminate all redundancies using normalization techniques because it cannot be assumed that all decompositions will be non-loss. Hence again, the decomposition of ***ACP* is a *lossy join decomposition*.**

So, the above tables are not in 5th normal form.

UNIT-4

UNIT-4

TRANSACTIONS:

A transaction is a unit of program execution that accesses and possibly updates various data items.

(or)

A transaction is an execution of a user program and is seen by the DBMS as a series or list of actions i.e., the actions that can be executed by a transaction includes the reading and writing of database.

Transaction Operations:

Access to the database is accomplished in a transaction by the following two operations,

1) **read(X)** : Performs the reading operation of data item X from the database.
2) **write(X)** : Performs the writing operation of data item X to the database.

Example:

Let T1 be a transaction that transfers $50 from account A to account B. This transaction can be illustrated as follows,

```
T1      : read(A);
          A := A – 50;
          write(A);
          read(B);
          B := B + 50;
          write(B);
```

Transaction Concept:

- The concept of transaction is the foundation for concurrent execution of transaction in a DBMS and recovery from system failure in a DBMS.
- A user writes data access/updates programs in terms of the high-level query language supported by the DBMS.
- To understand how the DBMS handles such requests, with respect to concurrency control and recovery, it is convenient to regard an execution of a user program or transaction, as a series of reads and writes of database objects.

- To read a database object, it is first brought in to main memory from disk and then its value is copied into a program. This is done by read operation.
- To write a database object, in-memory, copy of the object is first modified and then written to disk. This is done by the write operation.

Properties of Transaction (ACID):

There are four important properties of transaction that a DBMS must ensure to maintain data in concurrent access of database and recovery from system failure in DBMS.

The four properties of transactions are,

1) Atomicity
2) Consistency
3) Isolation
4) Durability

The acronym ACID is sometimes used to refer together to the four properties of transactions that we have presented here, Atomicity, Consistency, Isolation and Durability.

1) Atomicity: Users should be able to regard the execution of each transaction as atomic which means, either all actions of a transaction are carried out or none actions are carried out. That is, users should not have to worry about the effect of incomplete transactions when a system crash occurs.

Transaction can be incomplete for three kinds of reasons as follows:

i) First, a transaction can be aborted or terminated unsuccessfully by the DBMS because some changes arise during execution. If a transaction is aborted by the DBMS for some internal reason, it is automatically restarted and executed as a new transaction.

ii) Second, the system may crash because the power supply is interrupted, while one or more transactions are in progress.

iii) Third, a transaction may encounter an unexpected situation and decide to abort.

2) Consistency: Execution of a transaction in isolation (that is, with no other transaction executing concurrently at the same time) preserves the consistency of the database.

This property is the authority of the application programmer. That is, if the programmer wants some data to be consistent (remains same at any cost), then he gives the consistency permission to that data. So that, any transaction cannot change the consistent data. It can be verified easily that, if the database is consistent before an execution of the transaction, the database remains consistent after the execution of the transaction.

3) Isolation: Isolation property ensures that each transaction is unaware of other transactions executing concurrently in the system. Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions T_i and T_j, it appears that T_i is unaware of T_j i.e., whether T_j has started or finished or not and T_j is unaware of T_i . i.e., whether T_j has started or finished or not.

4) Durability: After a transaction completes successfully, the changes made to the database will be successful, even if the system failure occurs after that, the database remains safe.
The durability property guarantees that, once a transaction completes successfully, all the updates that are carried out on the database persists, even if there is a system failure after the transaction completes execution.

TRANSACTION STATES:

A transaction is seen by the DBMS as a series or list of actions. We therefore establish a simple transaction model named as transaction states.
A transaction must be in one of the following states,

- **Active State** : This is the initial state of a transaction. The transaction stays in this state while it is execution.
- **Partially Committed State** : This transaction state occurs after the final statement of the transaction has been executed.
- **Failed State** : This transaction state occurs after the discovery that normal execution can no longer proceed.

- **Aborted State**: This transaction state occurs after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.
- **Committed State** : This transaction state occurs after the successful transaction completion.

The transaction state diagram corresponding to a transaction is below,

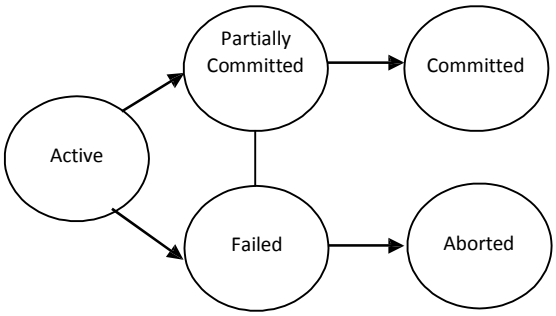


Fig: ***Transaction State Diagram***

A transaction starts in the active state. When it finishes statement, it enters the partially committed state. At this point, the transaction has completed its execution, but there is still a possibility that it may be aborted, since the actual output may still be temporarily residing in main memory and thus a hardware failure may stop its successful completion.

When the database system writes out successfully the output information to disk, then the transaction enters the committed state.

A transaction may also enters the failed state from the active state or from the partially committed state after the system determines that the transaction can no longer proceed with its normal execution because of hardware failure or logical errors. Such a transaction must be rolled back and the database has been restored to its state prior to the start of the transaction. Then, it is known as the aborted state. At this state, the system has two options as follows,

i) Restart the Transaction *:* It can restart the transaction, but only if the transaction was aborted as a result of some hardware failure or software error. A restarted transaction is considered to be a new transaction.

ii) ***Kill the Transaction*** *:* It can kill the transaction because of some internal logical error that can be corrected only by rewriting the application program or because the input was bad.

Serializability:

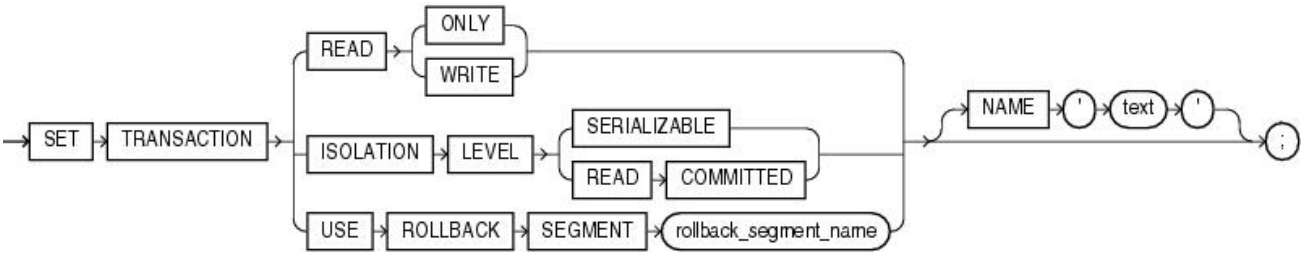
When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transaction are interleaved with some other transaction.

Schedule − A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.

Serial Schedule − It is a schedule in which transactions are aligned in such a way that one transaction is executed first. When the first transaction completes its cycle, then the next transaction is executed. Transactions are ordered one after the other. This type of schedule is called a serial schedule, as transactions are executed in a serial manner.

Read only Transaction

The SET TRANSACTION statement begins a read-only or read-write transaction, establishes an isolation level, or assigns the current transaction to a specified rollback segment. Read-only transactions are useful for running multiple queries against one or more tables while other users update the same tables.



If you want a transaction to be set as READ ONLY, you need to the transaction with the SET TRANSACTION READ ONLY statement. Note that a DML statement will start the transaction automatically. So you have to issue the SET TRANSACTION statement before any DML statements.

READ ONLY:

Establishes the current transaction as read-only, so that subsequent queries see only changes committed before the transaction began. The use of READ ONLY does not affect other users or transactions.

General Syntax:

```
SET TRANSACTION
{ { READ { ONLY | WRITE }
          | ISOLATION LEVEL
     { SERIALIZABLE | READ COMMITTED }
   | USE ROLLBACK SEGMENT rollback_segment
     } [ NAME string ]
         | NAME string
} ;
```

The code below shows a good example of READ ONLY transaction:

```
SQL> connect;
user name      : system
password       : database

SQL> SET TRANSACTION READ ONLY;
Transaction set.

SQL> SELECT * FROM fyi_links;
   ID  URL                  CREATED
-------  ----------------  ----------
101     fyicenter.com        07-may-06
110     centerfyi.com        07-may-06
112     oracle.com           07-may-06
113     sql.com              may-06
```

Restriction on Read-only Transactions:

Only the following statements are permitted in a read-only transaction:

i) Sub queries—SELECT statements without the for_update_clause

ii) LOCK TABLE

iii) SET ROLE

iv) ALTER SESSION

v) ALTER SYSTEM

Dirty Reads

Reading uncommitted data or WR conflicts or Temporary update problem is also called as Dirty Read. This problem occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed another transaction before it is changed back to its original value.

The first source of anomalies is that a transaction T_2 could read a database object 1 that has been just modified by another transaction T_1, which has not yet committed, such a read is called a dirty read.

Example:

Consider two transactions T_1 and T_2 where T_1 stands for transferring $100 from A to B and T_2 stands for incrementing both A and B by 6% of their accounts. Suppose that their actions are interleaved as follows,

1) T_1 deducts $100 from account A, then immediately,
2) T_2 reads accounts of A and B and adds 6% interest to each and then,
3) T_1 adds $100 to account B.

This corresponding schedule is illustrated as below,

T_1	T_2
Read(A)	
A:= A – 100	
Write(A)	
	Read(A)
	A:= A + 0.06 A
	Write(A)
	Read(B)
	B:= B + 0.06 B
	Write(B)
	Commit
Read(B)	
B:= B + 100	
Write(B)	
Commit	

The problem here is T_2 has added incorrect 6% interest to each A and B. Because before commitment that $100 is deducted from A, it has added 6% to account A and before commitment that $100 is credited to B, it has added 6% to account B. Thus, the result of this schedule is different from the result of the other schedule which is serializable first T_1, then T_2.

Other Isolation levels.

The ANSI/ISO SQL standard defines four levels of transaction isolation, with different possible outcomes for the same transaction scenario. That is, the same work performed in the same fashion with the same inputs may result in different answers, depending on your isolation level.

The four Isolation Levels are,

i) READ UNCOMMITTED

ii) READ COMMITTED

iii) REPEATABLE READ

iv) SERIALIZABLE

Oracle explicitly supports the READ COMMITTED and SERIALIZABLE isolation levels as they're defined in the standard. The SQL standard was trying to set up isolation levels that would permit various degrees of consistency for queries performed at each level. REPEATABLE READ is the isolation level that the SQL standard claims will guarantee a read-consistent result from a query. In the SQL standard definition, READ COMMITTED doesn't give you consistent results, and READ UNCOMMITTED is the level to use to get non blocking reads.

i) READ UNCOMMITTED:

The READ UNCOMMITTED isolation level allows dirty reads. Oracle Database doesn't use dirty reads, nor does it even allow them. The basic goal of a READ UNCOMMITTED isolation level is to provide a standards-based definition that allows for non blocking reads.

ii) READ COMMITTED:

The READ COMMITTED isolation level states that a transaction may read only data that has been committed in the database. There are no dirty reads (reads of uncommitted data). There may be non-repeatable reads.

READ COMMITTED is perhaps the most commonly used isolation level in database applications everywhere, and it's the default mode for Oracle Database. It's rare to see a different isolation level used in Oracle databases.

iii) REPEATABLE READ:

The goal of REPEATABLE READ is to provide an isolation level that gives consistent, correct answers and prevents lost updates. If you have REPEATABLE READ isolation, the results from a given query must be consistent with respect to some point in time. Most databases (not Oracle) achieve repeatable reads through the use of row-level, shared read locks. A shared read lock prevents other sessions from modifying data that you've read.

iv) SERIALIZABLE:

This is generally considered the most restrictive level of transaction isolation, but it provides the highest degree of isolation. A SERIALIZABLE transaction operates in an environment that makes it appear as if there are no other users modifying data in the database. Any row you read is assured to be the same upon a reread, and any query you execute is guaranteed to return the same results for the life of a transaction.

SERIALIZABLE does not mean that all transactions executed by users will behave as if they were executed one right after another in a serial fashion. It doesn't imply that there's some serial ordering of the transactions that will result in the same outcome.

Serializability

Serializability is a widely accepted standard that ensures the consistency of a schedule. A schedule is consistent if and only if it is serializable. A schedule is said to be serializable if the interleaved transactions produces the result, which is equivalent to the result produced by executing individual transactions separately.

Example:

Transaction T1	Transaction T2
read(X)	
write(X)	
read(Y)	
write(Y)	
	read(X)
	write(X)
	read(Y)
	write(Y)

Serial Schedule

Transaction T1	Transaction T2
read(X)	
write(X)	
	read(X)
	write(X)
read(Y)	
write(Y)	
	read(Y)
	write(Y)

Two interleaved transaction Schedule

The above two schedules produce the same result, these schedules are said to be serializable. The transaction may be interleaved in any order and DBMS doesn't provide any guarantee about the order in which they are executed.

There two different types of Serializability. They are,

i) Conflict Serializability

ii) View Serializability

i) Conflict Serializability:

Consider a schedule S1, consisting of two successive instructions IA and IB belonging to transactions TA and TB refer to different data items then it is very easy to swap these instructions.

The result of swapping these instructions doesn't have any impact on the remaining instructions in the schedule. If Ia and IB refers to same data item then the following four cases must be considered,

Case 1 : IA = read(x), IB = read(x),
Case 2 : IA = read(x), IB = write(x),
Case 3 : IA = write(x), IB = read(x),
Case 4 : IA = write(x), IB = write(x),

Case 1 : Here, both IA and IB are read instructions. In this case, the execution order of the instructions is not considered since the same data item x is read by both the transactions TA and TB.

Case 2 : Here, IA and IB are read and write instructions respectively. If the execution order of instructions is IA → I B, then transaction TA cannot read the value written by transaction TB in instruction IB. but order is IB → IA, then transaction TA can read the value written by transaction TB. Therefore in this case, the execution order of the instructions is important.

Case 3 : Here, IA and IB are write and read instructions respectively. If the execution order of instructions is IA → I B, then transaction TB can read the value written by transaction TA, but order is IB → IA, then transaction TB cannot read the value written by transaction TB. Therefore in this case, the execution order of the instructions is important.

Case 1 : Here, both IA and IB are write instructions. In this case, the execution order of the instructions doesn't matter. If a read operation is performed before the write operation, then the data item which was already stored in the database is read.

ii) View Serializability:

Two schedules S1 and S1' consisting of some set of transactions are said to be view equivalent, if the following conditions are satisfied,

1) If a transaction TA in schedule S1 performs the read operation on the initial value of data item x, then the same transaction in schedule S1' must also perform the read operation on the initial value of x.

2) If a transaction TA in schedule S1 reads the value x, which was written by transaction TB, then TA in schedule S1' must also perform the read the value x written by transaction TB.

3) If a transaction TA in schedule S1 performs the final write operation on data item x, then the same transaction in schedule S1' must also perform the final write operation on x.

Example:

Transaction T1	Transaction T2
read(x) x := x -10 write(x) read(y) y := y -10	 read(x) x := x *10 write(x)

write(y)	
	read(y) y := y / 10 write(y)

View Serializability Schedule S1

The view equivalence leads to another notion called view serializability. A schedule say S is said to be view Serializable, if it is view equivalent with the serial schedule.

Every conflict Serializable schedule is view Serializable but every view Serializable is not conflict Serializable.

CONCURRENCY CONTROL

In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions.

We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions.

Why DBMS needs a concurrency control?

In general, concurrency control is an essential part of TM. It is a mechanism for correctness when two or more database transactions that access the same data or data set are executed concurrently with time overlap. According to Wikipedia.org, if multiple transactions are executed serially or sequentially, data is consistent in a database. However, if concurrent transactions with interleaving operations are executed, some unexpected data and inconsistent result may occur. Data interference is usually caused by a write operation among transactions on the same set of data in DBMS. For example, the lost update problem may occur when a second transaction writes a second value of data content on top of the first value written by a

first concurrent transaction. Other problems such as the dirty read problem, the incorrect summary problem

Concurrency Control Techniques:

The following techniques are the various concurrency control techniques. They are:

concurrency control by Locks

Concurrency Control by Timestamps

Concurrency Control by Validation

1.CONCURRENCY CONTROL BY LOCKS:

LOCK: A lock is nothing but a mechanism that tells the DBMS whether a particular data item is being used by any transaction for read/write purpose.

There are two types of operations, i.e. read and write, whose basic nature are different, the locks for read and write operation may behave differently.

The simple rule for locking can be derived from here. If a transaction is reading the content of a sharable data item, then any number of other processes can be allowed to read the content of the same data item. But if any transaction is writing into a sharable data item, then no other transaction will be allowed to read or write that same data item.

Depending upon the rules we have found, we can classify the locks into two types.

Shared Lock: A transaction may acquire shared lock on a data item in order to read its content. The lock is shared in the sense that any other transaction can acquire the shared lock on that same data item for reading purpose.

Exclusive Lock: A transaction may acquire exclusive lock on a data item in order to both read/write into it. The lock is excusive in the sense that no other transaction can acquire any kind of lock (either shared or exclusive) on that same data item.

The relationship between Shared and Exclusive Lock can be represented by the following table which is known as Lock Matrix.

	hared	E xclusive
S		F

hared	RUE	ALSE
E xclusive	ALSE	F ALSE

TWO PHASE LOCKING PROTOCOL:

The use of locks has helped us to create neat and clean concurrent schedule. The Two Phase Locking Protocol defines the rules of how to acquire the locks on a data item and how to release the locks.

The Two Phase Locking Protocol assumes that a transaction can only be in one of two phases.

Growing Phase:

In this phase the transaction can only acquire locks, but cannot release any lock.

The transaction enters the growing phase as soon as it acquires the first lock it wants.

It cannot release any lock at this phase even if it has finished working with a locked data item.

Ultimately the transaction reaches a point where all the lock it may need has been acquired. This point is called Lock Point.

Shrinking Phase:

After Lock Point has been reached, the transaction enters the shrinking phase. In this phase the transaction can only release locks, but cannot acquire any new lock.

The transaction enters the shrinking phase as soon as it releases the first lock after crossing the Lock Point.

Two Phase Locking Protocol:

There are two different versions of the Two Phase Locking Protocol. They are:

Strict Two Phase Locking Protocol

Rigorous Two Phase Locking Protocol

In this protocol, a transaction may release all the shared locks after the Lock Point has been reached, but it cannot release any of the exclusive locks until the transaction commits. This protocol helps in creating cascade less schedule.

A Cascading Schedule is a typical problem faced while creating concurrent schedule. Consider the following schedule once again.

T1 T2

Lock-X (A)

Read A;

A = A - 100;

Write A;

Unlock (A)

Lock-S (A)

Read A;

Temp = A * 0.1;

Unlock (A)

Lock-X(C)

Read C;

C = C + Temp;

Write C;

Unlock (C)

Lock-X (B)

Read B;

B = B + 100;

Write B;

Unlock (B)

The schedule is theoretically correct, but a very strange kind of problem may arise here.

T1 releases the exclusive lock on A, and immediately after that the Context Switch is made.

T2 acquires a shared lock on A to read its value, perform a calculation, update the content of account C and then issue COMMIT. However, T1 is not finished yet. What if the remaining portion of T1 encounters a problem (power failure, disc failure etc) and cannot be committed?

In that case T1 should be rolled back and the old BFIM value of A should be restored. In such a case T2, which has read the updated (but not committed) value of A and calculated the value of C based on this value, must also have to be rolled back.

We have to rollback T2 for no fault of T2 itself, but because we proceeded with T2 depending on a value which has not yet been committed. This phenomenon of rolling back a child transaction if the parent transaction is rolled back is called Cascading Rollback, which causes a tremendous loss of processing power and execution time.

Using Strict Two Phase Locking Protocol, Cascading Rollback can be prevented.

In Strict Two Phase Locking Protocol a transaction cannot release any of its acquired exclusive locks until the transaction commits.

In such a case, T1 would not release the exclusive lock on A until it finally commits, which makes it impossible for T2 to acquire the shared lock on A at a time when A's value has not been committed. This makes it impossible for a schedule to be cascading.

RIGOROUS TWO PHASE LOCKING PROTOCOL:

In Rigorous Two Phase Locking Protocol, a transaction is not allowed to release any lock (either shared or exclusive) until it commits. This means that until the transaction commits, other transaction might acquire a shared lock on a data item on which the uncommitted transaction has a shared lock; but cannot acquire any lock on a data item on which the uncommitted transaction has an exclusive lock.

2.CONCURRENCY CONTROL BY TIMESTAMPS:

Timestamp ordering technique is a method that determines the serializability order of different transactions in a schedule. This can be determined by having prior knowledge about the order in which the transactions are executed.

Timestamp denoted by TS(TA) is an identifier that specifies the start time of transaction and is generated by DBMS. It uniquely identifies the transaction in a schedule. The timestamp of older transaction (TA) is less than the timestamp of a newly entered transaction (TB) i.e., TS(TA) < TS(TB).

In timestamp-based concurrency control method, transactions are executed based on priorities that are assigned based on their age. If an instruction IA of transaction TA conflicts with an instruction IB of transaction TB then it can be said that IA is executed before IB if and only if TS(TA) < TS(TB) which implies that older transactions have higher priority in case of conflicts.

Ways of generating Timestamps:

Timestamps can be generated by using,

i) System Clock : When a transaction enters the system, then it is assigned a timestamp which is equal to the time in the system clock.

ii) Logical Counter: When a transaction enters the system, then it is assigned a timestamp which is equal to the counter value that is incremented each time for a newly entered transaction.

Every individual data item x consists of the following two timestamp values,

i) WTS(x) (W-Timestamp(x)): It represents the highest timestamp value of the transaction that successfully executed the write instruction on x.

ii) RTS(x) (R-Timestamp(x)) : It represents the highest timestamp value of the transaction that successfully executed the read instruction on x.

TIMESTAMP ORDERING PROTOCOL:

This protocol guarantees that the execution of read and write operations that are conflicting is done in timestamp order.

Working of Timestamp Ordering Protocol:

The Time stamp ordering protocol ensures that any conflicting read and write operations are executed in time stamp order. This protocol operates as follows:

1) If TA executes read(x) instruction, then the following two cases must be considered,

i) TS(TA) < WTS(x)

ii) TS(TA) □ □WTS(x)

Case 1 : If a transaction TA wants to read the initial value of some data item x that had been overwritten by some younger transaction then, the transaction TA cannot perform the read operation and therefore the transaction must be rejected. Then the transaction TA must be rolled back and restarted with a new timestamp.

Case 2 : If a transaction TA wants to read the initial value of some data item x that had not been updated then the transaction can execute the read operation. Once the value has been

read, changes occur in the read timestamp value (RTS(x)) which is set to the largest value of RTS(x) and TS

2) If TA executes write(x) instruction, then the following two cases must be considered,

i) TS(TA) < RTS(x)

ii) TS(TA) < WTS(x)

iii) TS(TA) > WTS(x)

Case 1: If a transaction TA wants to write the value of some data item x on which the read operation has been performed by some younger transaction, then the transaction cannot execute the write operation. This is because the value of data item x that is being generated by TA was required previously and therefore, the system assumes that the value will never be generated. The write operation is thereby rejected and the transaction TA must be rolled back and should be restarted with new timestamp value.

Case 2 : If a transaction TA wants to write a new value to some data item x, that was overwritten by some younger transaction, then the transaction cannot execute the write operation as it may lead to inconsistency of data item. Therefore, the write operation is rejected and the transaction should be rolled back with a new timestamp value.

Case 3 : If a transaction TA wants to write a new value on some data item x that was not updated by a younger transaction, then the transaction can executed the write operation. Once the value has been written, changes occur on WTS(x) value which is set to the value of TS(TA).

Example:

T1	T2
read(y)	
	read(y)
	y:= y + 100
	write(y)
read(x)	
	read(x)
show(x+y)	
	x:= x – 100
	write(x)
	show(x+y)

The above schedule can be executed under the timestamp protocol when TS(T1) < TS(T2).

3. CONCURRENCY CONTROL BY VALIDATION

Validation techniques are also called as Optimistic techniques.

If read only transactions are executed without employing any of the concurrency control mechanisms, then the result generated is in inconsistent state.

However if concurrency control schemes are used then the execution of transactions may be delayed and overhead may be resulted. To avoid such issue, optimistic concurrency control mechanism is used that reduces the execution overhead.

But the problem in reducing the overhead is that, prior knowledge regarding the conflicting transactions will not be known. Therefore, a mechanism called “monitoring” the system is required to gain such knowledge.

Let us consider that every transaction TA is executed in two or three-phases during its life-time. The phases involved in optimistic concurrency control are,

1) Read Phase

2) Validation Phase and

3) Write Phase

1) Read phase: In this phase, the copies of the data items (their values) are stored in local variables and the modifications are made to these local variables and the actual values are not modified in this phase.

2) Validation Phase: This phase follows the read phase where the assurance of the serializability is checked upon each update. If the conflicts occur between the transaction, then it is aborted and restarted else it is committed.

3) Write Phase: The successful completion of the validation phase leads to the write phase in which all the changes are made to the original copy of data items. This phase is applicable only to the read-write transaction. Each transaction is assigned three timestamps as follows,

i) When execution is initiated I(T)

ii) At the start of the validation phase V(T)

iii) At the end of the validation phase E(T)

Qualifying conditions for successful validation:

Consider two transactions, transaction TA, transaction TB and let the timestamp of transaction TA is less than the timestamp of transaction TB i.e., TS (TA) < TS (TB) then,

1) Before the start of transaction TB, transaction TA must complete its execution. i.e., E(TA) < I(TB)

2) The values written by transaction TA must not be necessarily matched with the values read by transaction TB. TA must execute the write phase before TB initiate the execution of validation phase, i.e., I(TB) < E(TA) < V(TB)

3) If transaction TA starts its execution before transaction TB completes, then the write phase of transaction TB must be finished before transaction TA starts the validation phase.

Advantages:

i) The efficiency of optimistic techniques lie in the scarcity of the conflicts.
ii) It doesn't cause the significant delays.
iii) Cascading rollbacks never occurs.

Disadvantages:

i) Wastage in processing time during the rollback of aborting transactions which are very long.
ii) Hence, when one process is in its critical section (a portion of its code), no other process is allowed to enter. This is the principal of mutual exclusion.

RECOVERY

CRASH RECOVERY:

DBMS is a highly complex system with hundreds of transactions being executed every second. The durability and robustness of a DBMS depends on its complex architecture and its underlying hardware and system software. If it fails or crashes amid transactions, it is expected that the system would follow some sort of algorithm or techniques to recover lost data.

FAILURE CLASSIFICATION:

To see where the problem has occurred, we generalize a failure into various categories, as follows −

Transaction failure

A transaction has to abort when it fails to execute or when it reaches a point from where it can't go any further. This is called transaction failure where only a few transactions or processes are hurt.

Reasons for a transaction failure could be −

Logical errors − Where a transaction cannot complete because it has some code error or any internal error condition.

System errors − Where the database system itself terminates an active transaction because the DBMS is not able to execute it, or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.

System Crash

There are problems − external to the system − that may cause the system to stop abruptly and cause the system to crash. For example, interruptions in power supply may cause the failure of underlying hardware or software failure.

Examples may include operating system errors.

Disk Failure

In early days of technology evolution, it was a common problem where hard-disk drives or storage drives used to fail frequently.

Disk failures include formation of bad sectors, unreachability to the disk, disk head crash or any other failure, which destroys all or a part of disk storage.

Storage Structure

We have already described the storage system. In brief, the storage structure can be divided into two categories −

Volatile storage − As the name suggests, a volatile storage cannot survive system crashes. Volatile storage devices are placed very close to the CPU; normally they are embedded onto the chipset itself. For example, main memory and cache memory are examples of volatile storage. They are fast but can store only a small amount of information.

Non-volatile storage − These memories are made to survive system crashes. They are huge in data storage capacity, but slower in accessibility. Examples may include hard-disks, magnetic tapes, flash memory, and non-volatile (battery backed up) RAM.

Recovery and Atomicity

When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items. Transactions are made of various operations, which are atomic in nature. But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.

When a DBMS recovers from a crash, it should maintain the following −

It should check the states of all the transactions, which were being executed.

A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.

It should check whether the transaction can be completed now or it needs to be rolled back.

No transactions would be allowed to leave the DBMS in an inconsistent state.

There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction −

Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.

Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

LOG-BASED RECOVERY:

Log is a sequence of records, which maintains the records of actions performed by a transaction. It is important that the logs are written prior to the actual modification and stored on a stable storage media, which is failsafe.

Log-based recovery works as follows −

The log file is kept on a stable storage media.

When a transaction enters the system and starts execution, it writes a log about it.

<Tn, Start>

When the transaction modifies an item X, it write logs as follows −

<Tn, X, V1, V2>

It reads Tn has changed the value of X, from V1 to V2.

When the transaction finishes, it logs −

<Tn, commit>

The database can be modified using two approaches −

Deferred database modification − All logs are written on to the stable storage and the database is updated when a transaction commits.

Immediate database modification − Each log follows an actual database modification. That is, the database is modified immediately after every operation.

RECOVERY WITH CONCURRENT TRANSACTIONS

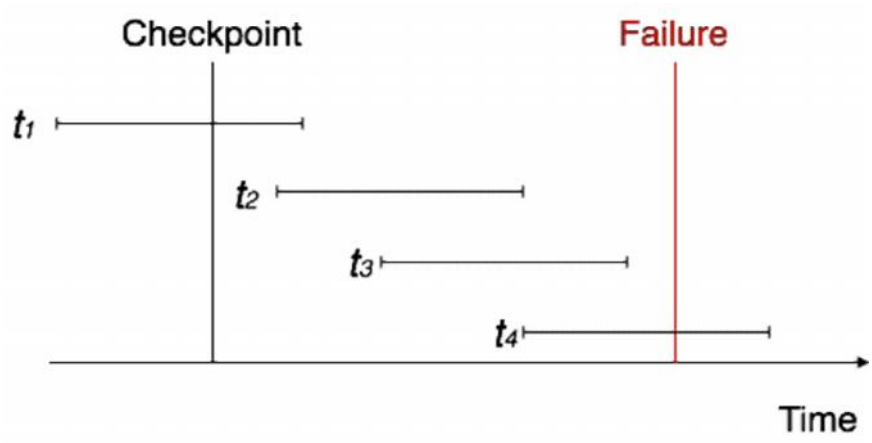
When more than one transaction are being executed in parallel, the logs are interleaved. At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering. To ease this situation, most modern DBMS use the concept of 'checkpoints'.

Checkpoint

Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system. As time passes, the log file may grow too big to be handled at all. Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

Recovery

When a system with concurrent transactions crashes and recovers, it behaves in the following manner −



The recovery system reads the logs backwards from the end to the last checkpoint.

It maintains two lists, an undo-list and a redo-list.

If the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>, it puts the transaction in the redo-list.

If the recovery system sees a log with <Tn, Start> but no commit or abort log found, it puts the transaction in undo-list.

All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

UNIT-5

UNIT-5

INTRODUCTION:

- **Primary Index** − Primary index is defined on an ordered data file. The data file is ordered on a **key field**. The key field is generally the primary key of the relation.

- **Secondary Index** − Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.

COMPARISION FILE ORGANIZATIONS:

As we know already, database consists of tables, views, index, procedures, functions etc. The tables and views are logical form of viewing the data. But the actual data are stored in the physical memory. Database is a very huge storage mechanism and it will have lots of data and hence it will be in physical storage devices. In the physical memory devices, these datas cannot be stored as it is. They are converted to binary format. Each memory devices will have many data blocks, each of which will be capable of storing certain amount of data. The data and these blocks will be mapped to store the data in the memory.

Any user who wants to view these data or modify these data, simply fires SQL query and gets the result on the screen. But any of these queries should give results as fast as possible. But how these data are fetched from the physical memory? Do you think simply storing the data in memory devices give us the better results when we fire queries? Certainly not. How is it stored in the memory, Accessing method, query type etc makes great affect on getting the results. Hence organizing the data in the database and hence in the memory is one of important topic to think about.

Types:

In a database we have lots of data. Each data is grouped into related groups called tables. Each table will have lots of related records. Any user will see these records in the form of tables in the screen. But these records are stored as files in the memory. Usually one file will contain all the records of a table.

As we saw above, in order to access the contents of the files – records in the physical memory, it is not that easy. They are not stored as tables there and our SQL queries will not work. We need some accessing methods. To access these files, we need to store them in certain order so that it will be easy to fetch the records. It is same as indexes in the books, or catalogues in the library, which helps us to find required topics or books respectively.

Storing the files in certain order is called file organization. The main objective of file organization is

- Optimal selection of records i.e.; records should be accessed as fast as possible.
- Any insert, update or delete transaction on records should be easy, quick and should not harm other records.
- No duplicate records should be induced as a result of insert, update or delete
- Records should be stored efficiently so that cost of storage is minimal.

There are various methods of file organizations. These methods may be efficient for certain types of access/selection meanwhile it will turn inefficient for other selections. Hence it is up to the programmer to decide the best suited file organization method depending on his requirement.

Some of the file organizations are

1. Sequential File Organization
2. Heap File Organization
3. Hash/Direct File Organization
4. Indexed Sequential Access Method
5. B+ Tree File Organization
6. Cluster File Organization

Let us see one by one on clicking the above links

Difference between Sequential, heap/Direct, Hash, ISAM, B+ Tree, Cluster file organization in database management system (DBMS) as shown below:

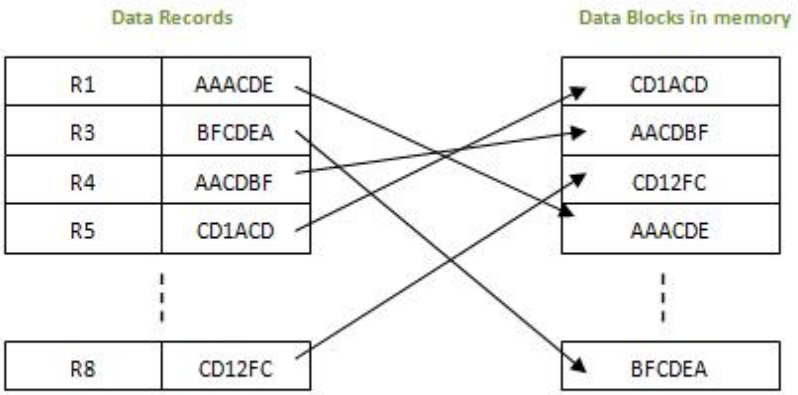
	Sequential	**Heap/Direct**	**Hash**	**ISAM**	**B+ tree**
Method of storing	Stored as they come or sorted as they come	Stored at the end of the file. But the address in the memory is random.	Stored at the hash address generated	Address index is appended to the record	Stored in a tre like structure
Types	Pile file and sorted file Method		Static and dynamic hashing	Dense, Sparse, multilevel indexing	

Design	Simple Design	Simplest	Medium	Complex	Complex
Storage Cost	Cheap (magnetic tapes)	Cheap	Medium	Costlier	Costlier
Advantage	Fast and efficient when there is large volumes of data, Report generation, statistical calculations etc	Best suited for bulk insertion, and small files/tables	Faster Access No Need to Sort Handles multiple transactions Suitable for Online transactions	Searching records is faster. Suitable for large database. Any of the columns can be used as key column. Searching range of data & partial data are efficient.	Searching rang of data & partia data are effi cient. No perfor manc degrades whe there is in sert delete / upd te. Grows an shrinks with data Works w ll i secondary storag devices and henc reducing di k I/C Since all da as ar at the leaf node searching is easy All data t lea node are sorte sequential linke list.
Disadvantage	Sorting of data each time for insert/delete/ update takes time and makes system slow.	Records are scattered in the memory and they are inefficiently used. Hence increases the memory size. Proper memory management is needed. Not suitable for	Accidental Deletion or updation of Data Use of Memory is inefficient Searching range of data, partial data, non-hash key column, searching single hash	Extra cost to maintain index. File reconstruction is needed as insert/update/delete. Does not grow with data.	Not suitab e fo static tables

		large tables.	column when multiple hash keys present or frequently updated column as hash key are inefficient.		

ISAM(INDEX SEQUENTIAL ACCESS METHOD):

This is an advanced sequential file organization method. Here records are stored in order of primary key in the file. Using the primary key, the records are sorted. For each primary key, an index value is generated and mapped with the record. This index is nothing but the address of record in the file.



In this method, if any record has to be retrieved, based on its index value, the data block address is fetched and the record is retrieved from memory.

Advantages of ISAM:

- Since each record has its data block address, searching for a record in larger database is easy and quick. There is no extra effort to search records. But proper primary key has to be selected to make ISAM efficient.

- This method gives flexibility of using any column as key field and index will be generated based on that. In addition to the primary key and its index, we can have index generated for other fields too. Hence searching becomes more efficient, if there is search based on columns other than primary key.

- It supports range retrieval, partial retrieval of records. Since the index is based on the key value, we can retrieve the data for the given range of values. In the same way, when a partial key value is provided, say student names starting with 'JA' can also be searched easily.

Disadvantages of ISAM:

- An extra cost to maintain index has to be afforded. i.e.; we need to have extra space in the disk to store this index value. When there is multiple key-index combinations, the disk space will also increase.

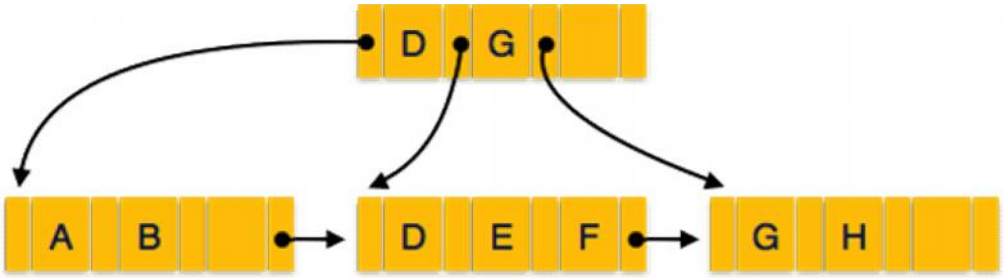
- As the new records are inserted, these files have to be restructured to maintain the sequence. Similarly, when the record is deleted, the space used by it needs to be released. Else, the performance of the database will slow down.

B^+ Tree:

A B^+ tree is a balanced binary search tree that follows a multi-level index format. The leaf nodes of a B^+ tree denote actual data pointers. B^+ tree ensures that all leaf nodes remain at the same height, thus balanced. Additionally, the leaf nodes are linked using a link list; therefore, a B^+ tree can support random access as well as sequential access.

Structure of B^+ Tree

Every leaf node is at equal distance from the root node. A B^+ tree is of the order**n** where **n** is fixed for every B^+ tree.



Internal nodes −

- Internal (non-leaf) nodes contain at least ⌈n/2⌉ pointers, except the root node.
- At most, an internal node can contain **n** pointers.

Leaf nodes −

- Leaf nodes contain at least ⌈n/2⌉ record pointers and ⌈n/2⌉ key values.
- At most, a leaf node can contain **n** record pointers and **n** key values.
- Every leaf node contains one block pointer **P** to point to next leaf node and forms a linked list.

B^+ Tree Insertion

- B^+ trees are filled from bottom and each entry is done at the leaf node.
- If a leaf node overflows −
    - Split node into two parts.
    - Partition at **i = ⌊(m+1)/2⌋.**

- First **i** entries are stored in one node.
- Rest of the entries (i+1 onwards) are moved to a new node.
- **i^{th}** key is duplicated at the parent of the leaf.

- If a non-leaf node overflows −
  - Split node into two parts.
  - Partition the node at **i = ⌈(m+1)/2⌉**.
  - Entries up to **i** are kept in one node.
  - Rest of the entries are moved to a new node.

B^+ Tree Deletion

- B^+ tree entries are deleted at the leaf nodes.
- The target entry is searched and deleted.
  - If it is an internal node, delete and replace with the entry from the left position.
- After deletion, underflow is tested,
  - If underflow occurs, distribute the entries from the nodes left to it.
- If distribution is not possible from left, then
  - Distribute from the nodes right to it.
- If distribution is not possible from left or from right, then
  - Merge the node with left and right to it.

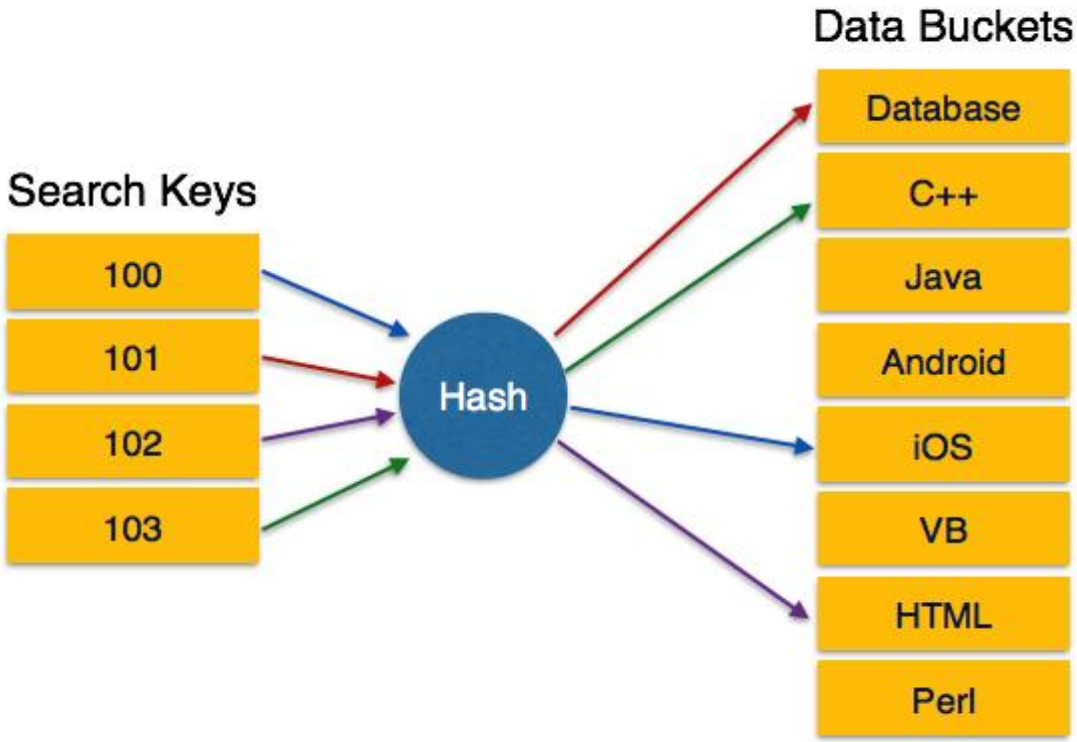
HASHING:

- **Bucket** − A hash file stores data in bucket format. Bucket is considered a unit of storage. A bucket typically stores one complete disk block, which in turn can store one or more records.

- **Hash Function** − A hash function, **h,** is a mapping function that maps all the set of search-keys **K** to the address where actual records are placed. It is a function from search keys to bucket addresses.

Static Hashing

In static hashing, when a search-key value is provided, the hash function always computes the same address. For example, if mod-4 hash function is used, then it shall generate only 5 values. The output address shall always be same for that function. The number of buckets provided remains unchanged at all times.



Operation

- **Insertion** − When a record is required to be entered using static hash, the hash function **h** computes the bucket address for search key **K**, where the record will be stored.

  Bucket address = h(K)

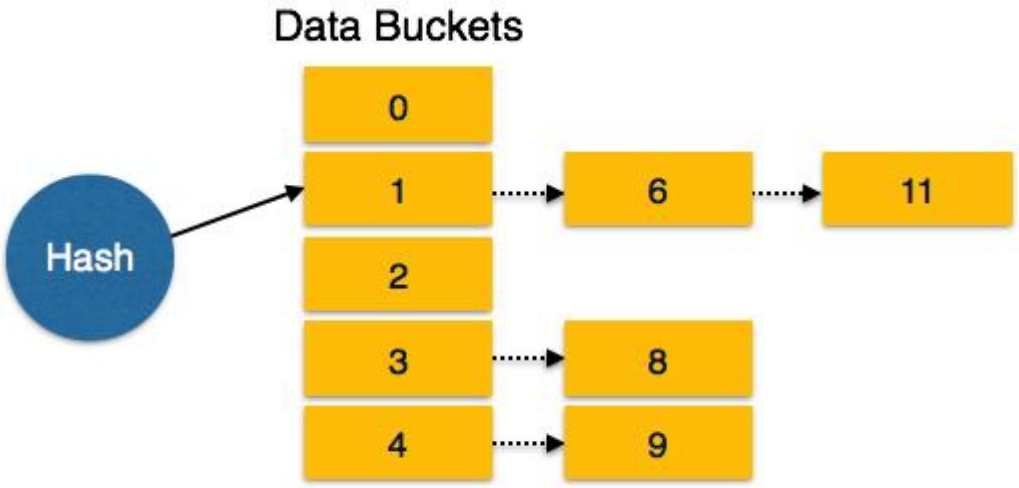
- **Search** − When a record needs to be retrieved, the same hash function can be used to retrieve the address of the bucket where the data is stored.

- **Delete** − This is simply a search followed by a deletion operation.

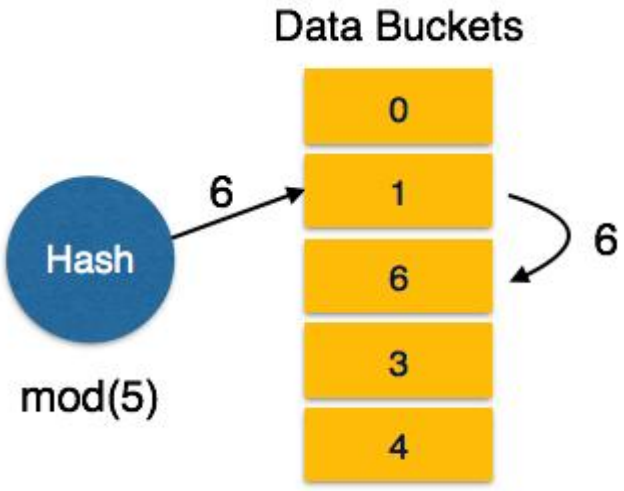
Bucket Overflow

The condition of bucket-overflow is known as **collision**. This is a fatal state for any static hash function. In this case, overflow chaining can be used.

- **Overflow Chaining** − When buckets are full, a new bucket is allocated for the same hash result and is linked after the previous one. This mechanism is called **Closed Hashing**.



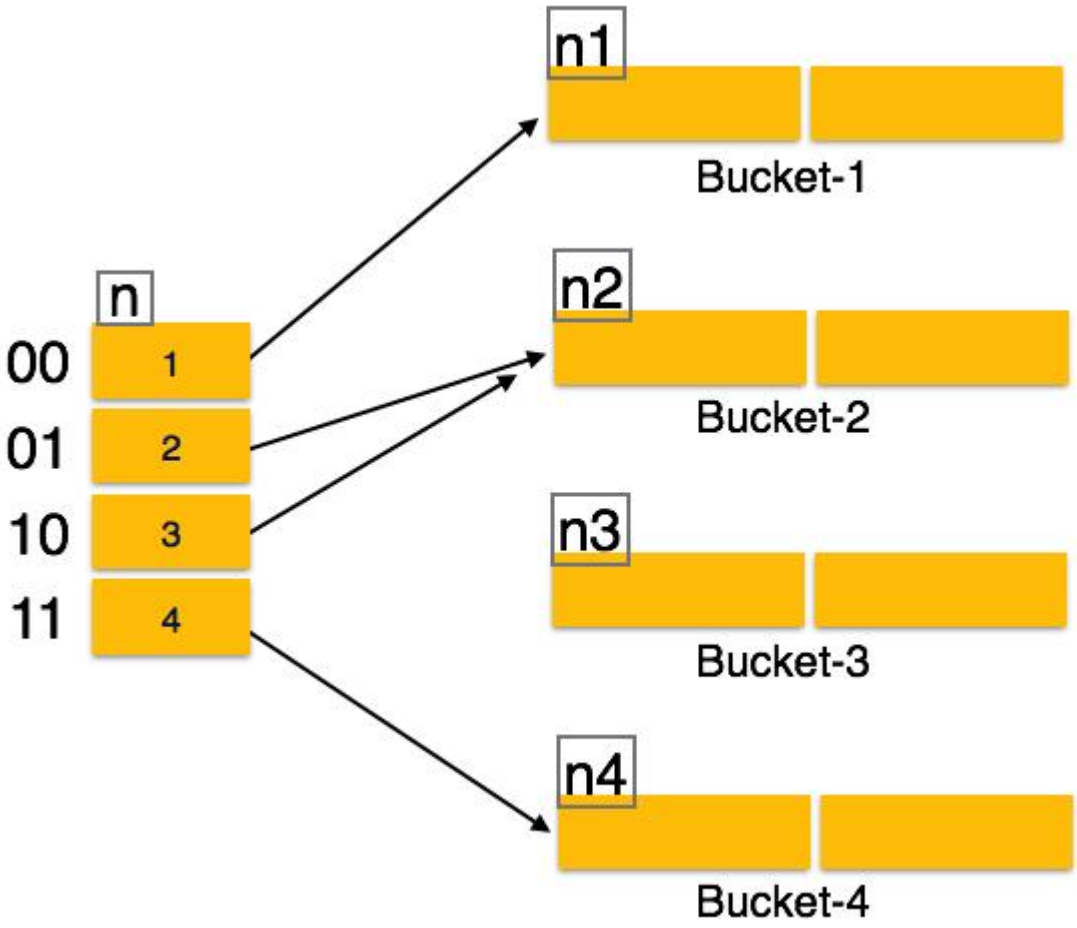
- **Linear Probing** − When a hash function generates an address at which data is already stored, the next free bucket is allocated to it. This mechanism is called **Open Hashing**.



Dynamic Hashing

The problem with static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks. Dynamic hashing provides a mechanism in which data buckets are added and removed dynamically and on-demand. Dynamic hashing is also known as **extended hashing**.

Hash function, in dynamic hashing, is made to produce a large number of values and only a few are used initially.



Organization

The prefix of an entire hash value is taken as a hash index. Only a portion of the hash value is used for computing bucket addresses. Every hash index has a depth value to signify how many bits are used for computing a hash function. These bits can address 2n buckets. When all these bits are consumed − that is, when all the buckets are full − then the depth value is increased linearly and twice the buckets are allocated.

Operation

- **Querying** − Look at the depth value of the hash index and use those bits to compute the bucket address.
- **Update** − Perform a query as above and update the data.
- **Deletion** − Perform a query to locate the desired data and delete the same.
- **Insertion** − Compute the address of the bucket

- If the bucket is already full.
    - Add more buckets.
    - Add additional bits to the hash value.
    - Re-compute the hash function.
- Else
    - Add data to the bucket,
- If all the buckets are full, perform the remedies of static hashing.

Hashing is not favorable when the data is organized in some ordering and the queries require a range of data. When data is discrete and random, hash performs the best.

Hashing algorithms have high complexity than indexing. All hash operations are done in constant time.

LINEAR HASHING:

Hash table is a data structure that associates keys with values. To know more about liner hashing refer Wikipedia. Here are main points that summarizes linear hashing.

- Full buckets are not necessarily split
- Buckets split are not necessarily full
- Every bucket will be split sooner or later and so all Overflows will be reclaimed and rehashed.
- Split pointer s decides which bucket to split s is independent to overflowing bucket

At level i, s is between 0 and 2i

s is incremented and if at end, is reset to 0.

hi (k)= h(k) mod(2i n)

hi+1 doubles the range of hi

Insertion and Overflow condition

Algorithm for inserting ‘k’:

1. b = h0(k)

2. if b < split-pointer then

3. b = h1(k)

Searching in the hash table for ‘k’:

1. b = h0(k)

2. if b < split-pointer then

3. b = h1(k)

4. read bucket b and search there

Example:

In the following M=3 (initial # of buckets)

Each bucket has 2 keys. One extra key for overflow.

s is a pointer, pointing to the split location. This is the place where next split should take place.

Insert Order: 1,7,3,8,12,4,11,2,10,13

After insertion till 12

When 4 inserted overflow occurred. So we split the bucket (no matter it is full or partially empty). And increment pointer.

So we split bucket 0 and rehashed all keys in it. Placed 3 to new bucket as (3 mod 6 = 3) and (12 mod 6 = 0). Then 11 and 2 are inserted. And now overflow. s is pointing to bucket 1, hence split bucket 1 by re- hashing it.

After split

Insertion of 10 and 13: as (10 mod 3 = 1) and bucket 1 < s, we need to hash 10 again using h1(10) = 10 mod 6 = 4th bucket.

When 13 is inserted same process is done, but it end up to the same bucket. But here is an overflow, we need to split 2nd bucket.

The final hash table contains s is moved to the top again as one cycle is completed. Now s will travel from 0 to 5th bucket, then 0 to 12, etc;